# Speech-To-Story: Automatizando a Geração de Histórias de Usuário

Gabriel Ferreira de Sousa
Instituto Federal da Bahia
Salvador, Brasil
Email: gabriel.ferreira@ifba.edu.br

Simone da Silva Amorim
Instituto Federal da Bahia
Salvador, Brasil
Email: simone.amorim@ifba.edu.br

***Abstract*—In the software development process, the requirements phase is the first one to be done. In this step, the most used technique to elicitate the system's requirements is the structured interview. On agile environments, with the objective to incorporate the manifest's premises, the requirements engineer then analyzes the collected answers and manually creates user stories, in order to support the development team. Despite achieving good results, this approach is prone to human mistakes such as in communication, ambiguities and incomplete requirements. Moreover, their impacts are only visible later, when validating the stories, since this creation is eventually only done later after the interview, resulting in wasted time and rework. As a solution, Speech-To-Story was developed: a mobile assistive solution to requirements elicitation, aiming to an automatic generation of stories and immediate validation. Questions are offered to the engineer, to lead a structured interview, and answers are collected. When finished, the solution analyzes the collected data and creates the stories, allowing validations and corrections still in interview-time.**

***Keywords*—CASE Tool, User Stories, Requirements, Interviews, Documentation.**

***Resumo*—No processo de desenvolvimento de software, a fase de requisitos é a primeira a ser executada. Nesta etapa, a técnica mais utilizada para levantar as premissas do sistema é a entrevista estruturada. Em ambientes ágeis, de modo a abarcar os preceitos do manifesto ágil, o engenheiro de requisitos então analisa as respostas obtidas e manualmente cria histórias de usuário, para dar suporte ao desenvolvimento. Embora produza bons resultados, esta abordagem está sujeita a falhas humanas de ambiguidade, comunicação e requisitos incompletos. Ademais, seus impactos são observáveis apenas posteriormente, quando da validação das histórias, pois sua criação eventualmente é realizada até dias após a entrevista, refletindo em tempo desperdiçado e re-trabalho. Como resolução, desenvolveu-se o Speech-To-Story: uma solução mobile assistiva para levantamento de requisitos, visando a geração automática de histórias e sua homologação imediata. São oferecidas perguntas para o engenheiro conduzir a entrevista estruturada, além da captura das respostas do cliente. À conclusão da conversa, a solução analisa as informações coletadas e cria as histórias, permitindo validações e correções em tempo de entrevista.**

***Palavras-chave*—Ferramenta CASE, História de Usuários, Requisitos, Entrevistas, Documentação.**

## I. Introdução

O processo para desenvolvimento de um *software* é definido por um conjunto de etapas, visando a criação e posterior evolução de um sistema, sob o escopo de um projeto [1]. A partir dos anos 1960 e 1970, com a crescente demanda por sistemas computacionais, observou-se que a maior parte destes projetos falhava em cumprir as expectativas do cliente, o orçamento e o tempo disponibilizado para sua produção [2]. Em resposta a esses problemas, advém a engenharia de *software*, utilizando princípios já consolidados de outras engenharias para nortear o contexto de sistemas computacionais, de modo a sistematizar, disciplinar e quantificar seu desenvolvimento, operação e manutenção [3] [4].

Os modelos definidos pela engenharia de *software* foram amplamente utilizados nas décadas seguintes e, com o passar do tempo, transformaram o processo de concepção de sistemas em algo rígido e inflexível, refletindo em dificuldades para atender às demandas frequentemente mutáveis dos clientes. Em reação a esse contexto, foi assinado o manifesto ágil [5], com foco em entregas rápidas e constantes do *software*.

Em ambientes ágeis, o uso de robustas documentações é desencorajado [5], já que a sua produção demanda recursos humanos e de tempo, os quais deveriam ser aplicados para o esforço de desenvolvimento do sistema em si. Como resultado, os diversos artefatos que antes eram escritos em cada fase do projeto passam a ser descartados ou reduzidos. Um exemplo que foi inviabilizado por essa premissa é o documento de requisitos, elaborado ainda durante a fase de levantamento de requisitos, com o intuito de determinar todas as características que o sistema a ser desenvolvido deve ter [1].

Como a abordagem do projeto passa a ser iterativa, de maneira a acomodar mudanças constantes nas exigências dos clientes, este tipo de documentação passa a ser substituído por histórias de usuário [6]. Nestas, os requisitos são convertidos em descrições acerca do sistema, na linguagem cotidiana do usuário final, apresentando como aquele *software* será manipulado.

Para criar as histórias de usuário, o engenheiro de requisitos realiza entrevistas com o cliente, utilizando as respostas que receber como matéria-prima para derivar estes documentos. Todo esse processo é feito de maneira *ad-hoc*, desde a concepção das perguntas usadas na entrevista, a captura das respostas, até análise do material e criação das histórias. Por conta desta condução manual, inclusive envolvendo interações interpessoais, este fluxo está sujeito a falhas humanas.

A solução proposta neste trabalho visa diminuir a intervenção manual na criação de histórias de usuário ao apresentar uma abordagem automatizada para gerá-las, através de uma

ferramenta assistiva. O Speech-To-Story se trata de uma aplicação *mobile* para uso em tempo de entrevista de levantamento de requisitos. Ela provê um conjunto de perguntas para que seu usuário, o engenheiro de requisitos, conduza a entrevista. Ao capturar e avaliar cada resposta do cliente, a solução identifica quais os atores que interagem com o sistema a ser desenvolvido, suas possíveis ações e as saídas que serão obtidas. Com estas informações, o Speech-To-Story produz histórias de usuário, permitindo validá-las imediatamente junto ao cliente.

Além desta Introdução, esta monografia está organizada como se segue. A seção II apresenta a movitação e justificativa. A seção III trata dos trabalhos e referências que embasaram o projeto. A seção IV detalha o Speech-To-Story, seus requisitos, funcionamento, arquitetura, fluxo da solução e sua distribuição. A seção V apresenta outras ferramentas correlatas a esta solução. A seção VI trata da validação utilizada. Finalmente, a seção VII trata da conclusão, limitações e trabalhos futuros.

## II. Motivação e Justificativa

A crise do *software*, ocorrida entre as décadas de 1960 e 1970, definiu um período em que o desenvolvimento de sistemas enfrentou dificuldades por conta da crescente demanda, falta de técnicas comprovadamente efetivas para orientar suas atividades, além da alta complexidade dos problemas que se desejava solucionar [2]. Com o intuito de atacar estas dificuldades, a engenharia de *software* surgiu, trazendo o uso de princípios já solidificados da engenharia para a criação de sistemas confiáveis e que funcionassem em máquinas reais [4]. A partir desta referência, objetivou-se desenvolver produtos com maior qualidade, dentro do prazo estipulado para o projeto e respeitando o orçamento estabelecido [7].

A engenharia de *software* definiu processos para padronizar a produção e manutenção dos seus produtos através de um fluxo sequencial organizado, contando com pontos de controle para verificar a evolução do desenvolvimento. Para atingir este fim, buscou-se definir uma ordem linear para as atividades que devem ser realizadas ao longo do processo de concepção do sistema, seus responsáveis e os artefatos que serão produzidos como resultado. Esta sequência de etapas é condensada em modelos de ciclo de vida.

Um dos modelos mais populares até a década de 1980 foi o cascata [8], o qual consistia de uma cadeia contendo as fases de levantamento de requisitos, sua análise, implementação, testes e implantação. Este modelo definia que uma nova fase só poderia ser iniciada após a conclusão da anterior, era altamente documentado, e impedia a existência de pontos de flexibilidade, não sendo permitido retroceder a uma etapa já realizada.

Como o objetivo do desenvolvimento do *software* é prover uma solução computacional a um problema manual, a disciplina de requisitos é a primeira a ser realizada neste processo, visando obter do cliente as características que o sistema deve ter e como ele deve ser utilizado [1]. Para elicitar as necessidades do usuário, executa-se tarefas de levantamento de requisitos e sua análise, prospectando e documentando os atributos do *software*. Como uma das saídas desta disciplina, é gerado o documento de casos de uso.

A rigidez dos modelos existentes induzia a situações em que o tempo gasto do levantamento de requisitos à entrega do sistema superasse a *deadline* estabelecida. Como resultado, os custos envolvidos em projetos de *software* também superavam as estimativas iniciais. Outro problema que este longo intervalo de tempo traz é que, por passar anos em desenvolvimento, os requisitos que o cliente desejava inicialmente poderiam mudar [9]. Estes fatores refletiram em projetos cancelados, e daqueles que foram concluídos, boa parte já não atendia as necessidades dos usuários quando entregues.

Diante desse cenário, as metodologias ágeis surgiram como uma reação a partir do seu manifesto [5], descrevendo os conceitos e práticas ágeis. Com o objetivo de entregar o *software* mais depressa, o processo de desenvolvimento passou a ter foco no código, diminuindo as documentações e usando uma abordagem iterativa para construir o sistema. Para atingir este propósito, as práticas ágeis defendem entregas menores e constantes dos serviços prioritários do sistema, visando assim ajustar rapidamente o projeto às novas demandas.

Os requisitos do cliente, portanto, são a base deste fluxo ágil. Neste cenário porém, a documentação de casos de uso tem dado lugar a histórias de usuário. Isso ocorre porque estes documentos são naturalmente detalhados, precisando se conhecer muitas informações do sistema antecipadamente [6], sendo a antítese dos preceitos ágeis de volatilidade. Como solução, utiliza-se histórias de usuário, contendo descrições menores e mais simples, gerando melhores resultados em ambientes de iterações constantes e incrementais [10].

Apesar da mudança na documentação utilizada, seu objetivo segue o mesmo: descrever os requisitos do cliente sobre o sistema em desenvolvimento. Para elicitar estas exigências e compor as histórias de usuário, a técnica mais amplamente utilizada é a entrevista com o cliente [11]. A prospecção de requisitos, portanto, é dependente da comunicação entre pessoas.

Como a entrevista se trata de uma conversa, sendo um processo interpessoal e verbal, ela está sujeita a inconsistências humanas, como expressão de requisitos ambíguos [12], incompletos [13], desconhecimento do entrevistador acerca do negócio do cliente [14], ou mesmo a falta de experiência do engenheiro de requisitos para escrever e conduzir uma entrevista [15] [16]. Estas imperfeições acarretam em riscos como a necessidade de re-trabalho, queda de qualidade do produto final [13], além de discrepância entre o desejo do cliente e o sistema entregue [15].

Algumas abordagens tentam atacar essas falhas através da padronização das entrevistas, como as entrevistas estruturadas. Nestas conferências, diretrizes e perguntas são definidas previamente, de modo que o engenheiro de requisitos conduza a conversa de maneira roteirizada [17]. Foi constatado que entrevistas estruturadas têm performance melhor do que aquelas sem estruturação [18]. Todavia, esta alternativa limita a liberdade do entrevistado [19], sendo desejável que haja flexibilidade para as perguntas e respostas.

Outra abordagem para diminuir falhas humanas em entrevistas é o uso de aplicações assistivas (CASE - Computer Aided Software Engineering). A utilização de sistemas CASE visa que o engenheiro de *software* utilize uma ferramenta sistêmica para auxiliá-lo a elucidar os requisitos durante a conversa com o cliente. Este recurso assistivo para o levantamento das exigências já é explorado na literatura [20] [21].

A solução detalhada neste trabalho apresenta uma aplicação CASE, desenvolvida para auxiliar no processo de levantamento de requisitos ao automatizar a criação de histórias de usuário, permitindo a validação imediata dos artefatos gerados. Para tanto, utiliza-se uma abordagem baseada em entrevistas estruturadas, provida por uma aplicação *mobile*, que deve ser usada no momento da entrevista de requisitos. Objetiva-se que a solução reduza possíveis falhas humanas ao gerar a documentação, melhorar a qualidade das histórias ao possibilitar sua validação imediata e permitir eventuais correções junto ao cliente já ao final da entrevista.

## III. Fundamentação Teórica

Antes de tratar da solução desenvolvida, faz-se necessário explanar os conceitos que foram utilizados como alicerce para a sua concepção, com o intuito de melhor compreendê-la. Desta forma, são apresentadas a seguir as bases teóricas sobre as quais o produto foi projetado e implementado.

### A. Métodos Ágeis

A origem dos métodos ágeis remonta à insatisfação com os custos envolvidos nos projetos de software dos anos 1980 e 1990, quando o tempo entre a definição de requisitos e a entrega do software ao cliente era muito longo. Como resposta, foi assinado o manifesto ágil, descrevendo os conceitos e práticas que norteiam os métodos ágeis, justificando essa metodologia e dividindo seus princípios em doze pontos [5].

Dentre essas diretrizes, definiu-se que nos métodos ágeis o foco está na implementação do código, ao invés do projeto, o que é alcançado ao utilizar uma abordagem iterativa para desenvolver o sistema em questão. O objetivo é entregar o *software* funcional em ciclos curtos, com incrementos rápidos e constantes para responder às novas exigências do cliente, que passa a participar ativamente da especificação e avaliação em cada iteração [7].

Até aquele momento, com o demorado processo de desenvolvimento e a constante mudança dos requisitos originais, diversos projetos eram cancelados, enquanto dos concluídos, boa parte não atendia mais às expectativas e necessidades do usuário final [9]. Isso se justifica pois os negócios dos clientes em questão estavam em constante mudança, sendo quase impossível produzir um conjunto de requisitos para o sistema que também não mudasse desde a fase de levantamento de requisitos até a entrega do *software*, como pedia o modelo cascata [8]. Concluiu-se assim que o produto precisava também evoluir rapidamente, de modo a refletir as alterações do negócio, reduzir os custos do processo e re-trabalho [5]

Para possibilitar esse desenvolvimento mais acelerado, a especificação, projeto e implementação são intercalados. Isso faz com que, nas metodologias ágeis, a documentação seja mínima, e os requisitos sejam definidos de maneira parcial a cada iteração [5]. É para dar suporte a esse contexto que a solução apresentada neste trabalho foi desenvolvida.

### B. Requisitos

A motivação do processo de desenvolvimento de um novo sistema ou funcionalidade decorre da ideia de resolver, de modo sistêmico, as necessidades de um determinado cliente [7]. Com o intuito de descrever como o projeto solucionará essas carências, os requisitos de um *software* definem os serviços que ele deve oferecer. Estes requisitos podem ser funcionais, detalhando como o sistema deve se comportar frente às regras do negócio, ou não-funcionais, que tratam das características de qualidade do produto [7].

Como a finalidade dos requisitos é descrever o comportamento do que ainda será concebido, eles estão, portanto, ligados às primeiras etapas da construção do sistema. Esta premissa está descrita na metodologia do Rational Unified Process (RUP), um representante da abordagem tradicional de desenvolvimento, que divide todo o processo de produção de *software* em fases sequenciais. A primeira dessas etapas é a de concepção, que define casos de uso, identifica os atores que interagem com o sistema e o nível desta interação [1].

A Figura 1 evidencia a fases definidas no RUP, conforme a divisão supracitada. Nesta imagem, a disciplina de requisitos se inicia na concepção e se solidifica ao longo da elaboração. A disciplina de Requisitos, assim como as demais, é um *workflow*, representando uma sequência de atividades que produzem um resultado de valor observável [1].

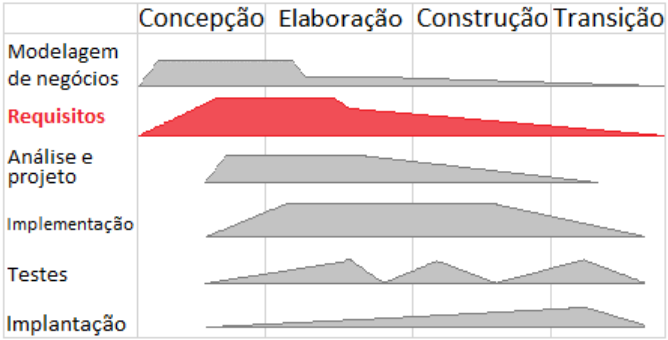


Figura 1: Disciplinas do RUP [1].

Esta sequência de atividades da disciplina de requisitos é observável na Figura 2: inicia-se ao analisar o problema do cliente e/ou entender suas necessidades, dependendo se este problema já é resolvido por um *software* existente ou se é preciso criar um novo. A partir desta análise, deve-se então definir o sistema, isto é, como o produto vai resolver o problema apresentado. Com esta definição, deve-se gerenciar o escopo coberto por este sistema para entender se ele conseguirá englobar aquele problema, de modo a antecipar se este será capaz de solucionar aquela necessidade por completo. Caso positivo, sendo este escopo realizável, refina-se a definição do sistema para incluir aquela nova exigência nos requisitos do

desenvolvimento. Caso contrário, se for impossível realizar, então deve-se voltar a entender as necessidades do cliente para dividir esta exigência em mais requisitos, com escopos menores, que sejam possíveis de desenvolver.

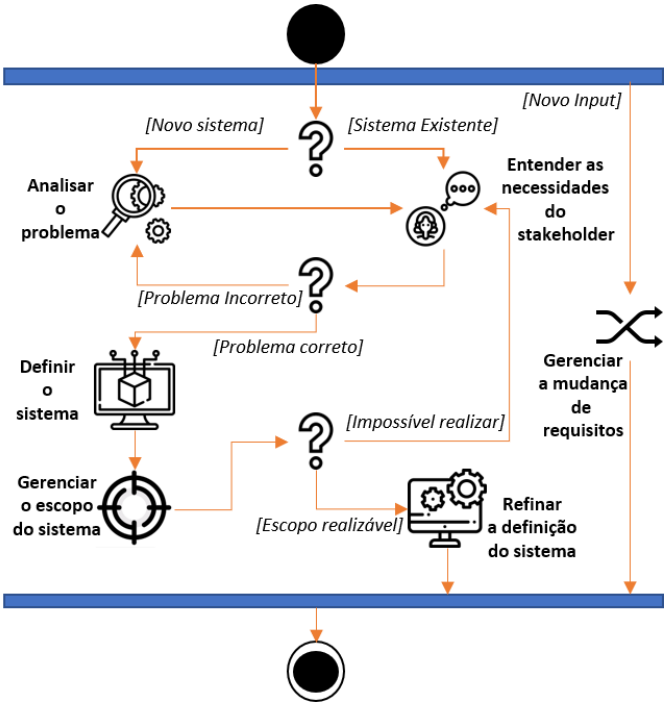


Figura 2: Workflow da disciplina de Requisitos [1].

Ainda segundo o RUP, ao final desta disciplina de Requisitos é criado um documento de visão, identificando atores, usuários e sistemas externos que interagem com aquele a ser desenvolvido. Também devem estar representados os casos de uso, explicando como o sistema se comporta e interage, passo a passo, com os atores. Este mesmo modelo de casos de uso é manipulado durante diversos momentos do desenvolvimento, como levantamento de requisitos, análise e projeto, e até mesmo testes [1].

### C. Histórias de usuário

Nas abordagens de desenvolvimento tradicionais, com o intuito de organizar os requisitos levantados, são criados documentos de caso de uso, contendo narrativas detalhadas para explicar como o usuário interage e utiliza o *software* [1]. Com a adoção de metodologias ágeis, constatou-se que esta documentação induz a necessidade de se saber muito acerca do sistema antecipadamente, o que configura uma antítese aos preceitos ágeis, culminando em uma reação através da adoção de histórias de usuário [6].

Histórias de usuário têm o mesmo objetivo dos casos de uso: elas descrevem funcionalidades do sistema que são importantes para o cliente ou usuário final [22], apresentando as ações que aquela pessoa executa, a partir do seu próprio ponto de vista. Diferente do documento tradicional, são frases simples e curtas, sem detalhamento, servindo para dar suporte a atividades iterativas e incrementais [10].

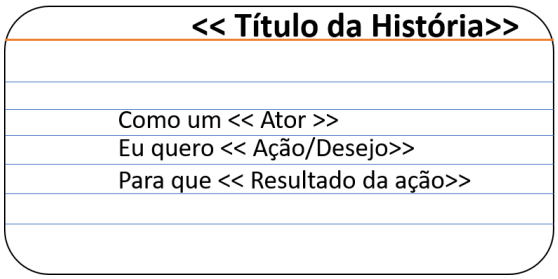


Figura 3: Exemplo de história de usuário, em um *Story Card*.

Histórias de usuário dividem os detalhados casos de uso em frases menores, apresentando-as em pequenos cartões chamados *story card*, de modo a sintetizar cada funcionalidade [10], conforme exemplificado na Figura 3. Se porventura o caso de uso não couber em um único cartão, aquele requisito deve ser refinado e dividido em outros mais curtos, uma vez que cada história deve ser pequena, independente, negociável, estimável e testável [23].

Como vantagem de seu uso, histórias de usuário enfatizam a comunicação verbal ao invés da escrita, são facilmente compreendidas, tem o tamanho correto para ajudar no planejamento e iteração, além de promover um entendimento mais fácil sobre o sistema [10].

### D. Entrevistas

Para que os requisitos do sistema sejam coletados de maneira efetiva, é definido na literatura que deve-se concentrar esforços ao longo do processo de levantamento de requisitos [13]. Dentre as técnicas para a elucidação de requisitos a mais utilizada é a entrevista, identificando opiniões e fatos sobre o sistema a partir de conversas face-a-face com o usuário e demais *stakeholders* [24].

Apesar de comumente empregada, esta abordagem está sujeita a falhas, uma vez que a entrevista consiste da transferência de conhecimento e ideias entre pessoas. É o caso de possíveis ambiguidades [12], desconhecimento do entrevistador sobre como escrever e conduzir entrevistas [15] [16], além de falta de entendimento do deste sobre o negócio do cliente [14].

Caso uma destas falhas ocorra e não se detecte, podem-se originar problemas posteriores ao longo do projeto, tais como: desnecessárias melhorias de curto prazo [16], discrepância entre o que o cliente quer e o produto final [15], re-trabalho e queda de qualidade do produto [13].

Para sanar estas falhas, mitigar riscos e produzir resultados melhores, a literatura define entrevistas estruturadas [18]. Neste tipo de entrevista, o engenheiro de requisitos conduz a conversa utilizando um conjunto pré-definido de questões, apenas buscando respostas para elas, seguindo diretrizes [17].

Para facilitar a realização deste tipo de entrevista, existem tecnologias que podem auxiliar o entrevistador. Estas são ferramentas assistivas, a partir de inteligências auxiliares, que oferecem roteiros já estabelecidos [21].

### E. CASE

Uma ferramenta CASE (computer Aided Software Engineering) é um termo genérico usado para denotar qualquer forma de suporte automatizado para engenharia de *software* [25]. Intenciona-se, com seu uso, melhorar a qualidade dos processos ao mecanizar rotinas, economizando trabalho e eliminando uma fonte de erro humano [26].

O uso destes sistemas auxiliares no contexto de engenharia não é recente, e induz benefícios como a redução de esforços entre 30% e 40%, refinamento na qualidade do *software* e na consistência das documentações, além de uma mudança na cultura das empresas para uma abordagem mais estruturada [25]. Estas ferramentas assistivas também podem ser utilizadas durante entrevistas, em conjunto com perguntas pré-definidas, refletindo em melhorias na produtividade [20] [21].

A solução apresentada neste trabalho busca atuar como uma ferramenta assistiva e ofertar uma abordagem estruturada para a entrevista de levantamento de requisitos. Para isso, a aplicação cria as perguntas que devem ser realizadas pelo entrevistador e, a partir da análise das respostas obtidas, formula novas perguntas. Este fluxo só é interrompido quando o próprio usuário indicar que não existem mais elementos do sistema a se identificar.

### F. Desenvolvimento Mobile e Android

Após décadas de seu lançamento e 11 versões em produção, a plataforma Android detém cerca de 74% do mercado dos *smartphones* [27]. Esta base de usuários traz também o interesse ao desenvolvimento de aplicações para o sistema, com sua loja ultrapassando 2.56 milhões de aplicativos disponíveis [28].

Os *software* que executam nesta plataforma, porém, possuem restrições em função da limitada disponibilidade de recursos de *hardware* dos dispositivos, como memória e armazenamento. Por outro lado, a portabilidade dos aparelhos, seus sensores e o microfone permitem a criação de aplicações que se aproveitem destes recursos. O desenvolvimento de aplicativos para Android pode ser realizado através de linguagens como Java e o Kotlin. Por outro lado, alternativas como C# e Xamarin produzem tanto aplicações nativas quanto híbridas com a plataforma iOS [29].

Um desafio em comum para as aplicações nesta plataforma, independente da linguagem escolhida, é a persistência de dados coletados durante a execução do *software*. Existem soluções que armazenam estas informações em servidores remotos, tornando a aplicação local *stateless*, mas ao mesmo tempo exigindo conexão com internet para envio e recebimento dos dados. Alternativas são o armazenamento local, utilizando escrita e leitura de arquivo, e bancos de dados adaptados para necessidades *mobile*, como SQLite, Oracle Berkeley DB e Interbase.

Para compor a solução proposta visando a execução em dispositivos Android, foi escolhida a combinação de C# e Xamarin, com persistência local via SQLite [30]. O intuito é recriar, em plataformas móveis, a abordagem CASE que auxilia durante o levantamento de requisitos, visto que esta até então se concentra em aplicações *desktop*. Objetiva-se com isso ampliar as capacidades das ferramentas CASE ao se beneficiar dos recursos oferecidos pelos dispositivos *mobile*, tais como a portabilidade dos aparelhos, seu crescente poder computacional, a possibilidade de acesso a servidores e APIs remotas via internet móvel, além de seus equipamentos de *hardware* embutidos como o microfone.

A combinação das tecnologias escolhidas supracitadas promove a separação entre as camadas de interface gráfica e lógica. A primeira é definida por arquivos AXML, contendo elementos gráficos do Android como botões, listas, *toolbars* e *cards*. A segunda, por características funcionais em C# como classes de *controller*, gerenciando o funcionamento interno da aplicação, e de atividade, responsáveis por iniciar a interface gráfica e permitir o *input* e *output* do usuário. De posse dessas informações, esta segunda camada também define classes de modelo para enviar requisições à plataformas externas, ou armazená-las no banco de dados local. Este fluxo é observável na Figura 4.

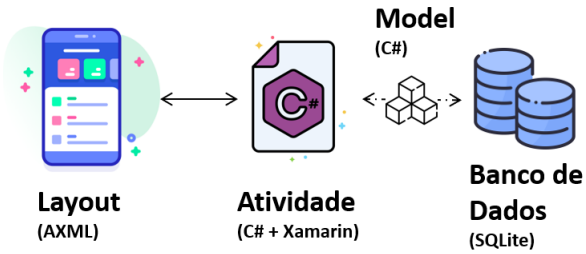


Figura 4: Estrutura das funcionalidades com C# e Xamarin.

### G. Processamento de Linguagem Natural

Dentre as sub-áreas de inteligência artificial, o processamento de linguagem natural (NLP - em inglês) trata de fornecer meios aos computadores de entender textos em linguagem humana [31] [32]. Através da NLP, uma inteligência é capaz de realizar tarefas como analisar sentimentos, aprender contextos e analisar a sintaxe e morfologia de palavras.

A partir desta inteligência, produtos podem ser ofertados e incorporados em outros sistemas. Através da análise morfológica, por exemplo, é possível que um programa classifique cada palavra presente em uma frase conforme a sua classe gramatical, como substantivo, pronome, adjetivo ou verbo [33]. É possível também graduar a precisão com que esta avaliação foi realizada, provendo em sequência de mais provável a menos provável, as demais possibilidades morfológicas para aquela palavra.

Permitindo a integração destas funcionalidades aos desenvolvedores em geral, a Apache disponibiliza o OpenNLP, uma biblioteca de código aberto para processamento de linguagem natural [34], oferecendo suas funções de NLP em português, inglês e outros sete idiomas. É possível implementar essas funcionalidades diretamente em código, importando esta *library* e seus binários para treinamento da inteligência artificial.

### H. Correção Gramatical

Ferramentas de correção gramatical têm como objetivo avaliar possíveis erros de digitação, concordância, tempos verbais e ordem morfológica em uma determinada frase. Este é um tipo de avaliação que difere da análise morfológica, pois busca detectar possíveis erros em uma frase completa. Esta última, por sua vez, apenas classifica cada palavra quanto à sua classe, não emitindo valor sobre sua colocação na frase [35].

Para implementar este tipo de ferramenta em código, existem *web services* de análise gramatical como o TextGears [1], GrammarBot [2], Grammarly [3] e a API de checagem gramatical da Microsoft [4]. Todas elas expõem APIs REST e seguem um fluxo similar: como entrada, essas APIs recebem a frase que se quer avaliar, através de uma requisição GET. Como resposta, o serviço envia um JSON, indicando os erros gramaticais encontrados na frase. Um exemplo ilustrando o resultado de uma avaliação gramatical está no Código 1, em que se realiza uma requisição para o TextGears.

Neste exemplo, analisa-se a seguinte frase propositalmente incorreta: "Como um estidante, eu gostaria de acessar o sistema para obter minhas notas". O serviço irá apontar quais palavras estão incorretas, as suas posições na sentença e uma lista de sugestões para consertá-las, ordenadas da mais provável de ser a correta para a menos provável.

```
{
  "response": {
    "errors": [
      {
        "id": "e525677210",
        "offset": 9,
        "length": 9,
        "description": {
          "en": "Possivel erro ortografico
                 encontrado"
        },
        "bad": "estidante",
        "better": [
          "estudante",
          "estiante",
          "estirante",
          "estilante",
          "esticante",
          "estimante",
          "estivante"
        ],
        "type": "spelling"
      }
    ]
  }
}
```

Código 1: Retorno da correção gramatical do TextGears.

[1]TextGears: https://textgears.com/api/
[2]GrammarBot: https://www.grammarbot.io
[3]Grammarly: https://www.grammarly.com
[4]API de checagem gramatical da Microsoft: https://docs.microsoft.com/en-us/bing/search-apis/bing-spell-check/reference/endpoints

## IV. Solução Desenvolvida

Em consonância a todos os elementos de fundamentação teórica expostos, desenvolveu-se uma solução aplicável a ambientes ágeis, com o intuito de auxiliar no processo de levantamento de requisitos, através da criação de documentação de maneira automática. Se trata de uma ferramenta CASE como uma aplicação *mobile* para plataforma Android, usada no momento da entrevista de levantamento de requisitos.

### A. Descrição da solução

A solução desenvolvida se chama Speech-To-Story, uma aplicação mobile para realizar a geração automática de histórias de usuário. Seu objetivo é auxiliar o engenheiro de requisitos ao promover uma abordagem de entrevista estruturada, usando capacidades de dispositivos móveis como seu microfone e armazenamento interno. Para isso, a solução fornece perguntas roteirizadas em sequência, captação das respostas faladas pelo entrevistado, sua análise e avaliação dos dados obtidos.

Este sistema conta com uma camada de processamento de linguagem natural, avaliando morfologicamente cada palavra das respostas obtidas junto ao cliente entrevistado. Com o resultado desta análise, gera-se a documentação dos requisitos em formato de histórias de usuário, com vista ao seu uso posterior ao longo do processo de desenvolvimento.

A aplicação também atua como um gerenciador de histórias de usuário, sendo possível gerir aquelas já criadas através de funções como clonagem, edição ou remoção, além de ser possível criar mais histórias manualmente. Também é disponibilizada uma avaliação gramatical sobre as histórias criadas, oferecendo sugestão de correções, quando detectados erros nas frases. Por fim, a ferramenta oferta a possibilidade de exportação das histórias.

Ao automatizar ações do processo de entrevista e documentação, esta solução visa diminuir a intervenção manual de modo a mitigar possíveis falhas humanas durante o levantamento das funcionalidades. Esta abordagem móvel também objetiva trazer dinamicidade ao processo de criação de histórias de usuário para que, já ao fim da entrevista, seja possível apresentar as histórias ao entrevistado para validação imediata. Finalmente, ela oferece flexibilidade para manutenção das histórias, de modo que o engenheiro de requisitos pode editar cada requisito obtido conforme o que o cliente realmente desejar.

### B. Requisitos da Solução

Os requisitos de um *software* descrevem os serviços que o produto deve prover, bem como suas restrições. Eles determinam como este sistema deve se comportar, assim como as propriedades que deve conter [7]. É a partir destes atributos que se definem as métricas usadas como critério de aceitação de uma dada solução [36].

Estes requisitos podem ser separados entre funcionais e não-funcionais, buscando descrever diferentes características: o primeiro descreve como o sistema deve reagir, e quais ações ele deve ser capaz de executar, a partir de uma determinada

entrada. O segundo, por outro lado, define características inerentes ao próprio sistema, voltadas especificamente aos seus próprios serviços [7].

*1) Requisitos Funcionais:* A solução proposta contém uma série de requisitos funcionais, cuja completude é necessária para definir se o sistema foi capaz de cumprir os objetivos originalmente propostos. A lista de requisitos funcionais é apresentada na Tabela I:

| ID | Requisito Funcional | Ator |
|---|---|---|
| RF1 | O sistema deve ser capaz de criar e remover projetos de desenvolvimento de *software* | Usuário |
| RF2 | O sistema deve ser capaz de criar e remover entrevistas de levantamento de requisitos | Usuário |
| RF3 | O sistema deve prover perguntas para coleta de informações ao seu usuário | Usuário |
| RF4 | O sistema deve permitir capturar *inputs* de voz de seu usuário | Usuário |
| RF5 | O sistema deve ser capaz de transcrever *inputs* de voz em texto | Sistema |
| RF6 | O sistema deve ser capaz de analisar respostas de uma entrevista para definir e gerar novas perguntas | Sistema |
| RF7 | O sistema deve permitir a validação e correção dos conteúdos que transcrever | Usuário |
| RF8 | O sistema deve ser capaz de classificar a completude de uma entrevista a partir de suas perguntas e respostas realizadas | Sistema |
| RF9 | O sistema deve ser capaz de classificar morfologicamente as palavras de um texto | Sistema |
| RF10 | O sistema deve permitir a geração e exibição de histórias de usuário a partir das respostas coletadas na entrevista | Usuário |
| RF11 | O sistema deve ser capaz de permitir a validação das histórias geradas junto ao cliente | Usuário |
| RF12 | O sistema deve permitir editar, clonar, remover ou adicionar histórias junto àquelas previamente geradas | Usuário |
| RF13 | O sistema deve ser capaz de exportar as histórias de usuário de um determinado projeto | Usuário |

Tabela I: Requisitos funcionais da solução proposta.

*2) Requisitos Não-Funcionais:* A solução também possui com uma lista de requisitos não-funcionais, apresentada na Tabela II e categorizado de acordo com Sommerville [7].

| ID | Requisito Não-Funcional | Categoria |
|---|---|---|
| RNF1 | A solução deve ser executada em um *smartphone* Android, versão 8.1 ou superior | Ambiental |
| RNF2 | Respostas obtidas para um dado ator não podem ser reaproveitadas para outro ator | Proteção |
| RNF3 | Histórias de usuário geradas em uma entrevista não podem ser exibidas nem manipuladas em outra entrevista | Proteção |
| RNF4 | O sistema não pode limitar a quantidade máxima de perguntas e respostas de uma determinada entrevista | Operacional |
| RNF5 | O sistema não pode limitar a quantidade máxima de histórias geradas em cada entrevista | Operacional |
| RNF6 | A solução deve implementar processamento de linguagem natural usando a biblioteca OpenNLP | Desenvolvimento |
| RNF7 | A solução será desenvolvida utilizando o framework Xamarin e persistência via banco de dados SQLite | Desenvolvimento |
| RNF8 | A solução deve implementar avaliação para correções gramaticais através de requisições REST a sistemas externos | Desenvolvimento |
| RNF9 | Todo o processamento para geração de histórias deve ocorrer de maneira local, naquele mesmo *smartphone* | Ambiental |
| RNF10 | A solução deve armazenar os dados coletados durante sua execução apenas em banco de dados local, naquele mesmo *smartphone* | Ambiental |
| RNF11 | A solução deve permitir exportação de histórias em formatos de texto e imagem | Operacional |

Tabela II: Requisitos não-funcionais da solução proposta.

### C. Funcionamento da Solução

Para que a solução opere, o engenheiro de requisitos deve estar com a aplicação do Speech-To-Story instalada e executando em primeiro plano, em um aparelho Android na versão 8.1 ou superior. Com o aplicativo aberto em sua tela inicial, deve-se criar um projeto sobre o qual se quer trabalhar, ou clicar em um que já esteja criado. Independente da escolha, uma vez dentro do projeto, uma de suas entrevistas será manipulada. Pode-se optar por iniciar uma nova entrevista, vazia, ou dar sequência a alguma que tenha sido previamente interrompida.

Ao iniciar uma nova entrevista, o Speech-To-Story primeiro oferecerá perguntas de ordem genérica, com o intuito de obter informações gerais do cliente, como a empresa em que trabalha, seu negócio, quem ele é e sobre o que se trata aquele sistema. Estas são perguntas opcionais, e não vão interferir no fluxo da solução, mas servem para aclimatar o entrevistado à ferramenta e como ela funciona.

Para de fato começar o fluxo que gera histórias, a aplicação então oferece questões que visam identificar os atores que vão interagir com o sistema do cliente. Estas perguntas devem ser realizadas pelo engenheiro de requisitos durante a entrevista, de modo a obter um retorno do entrevistado. Antes que este comece a falar sua resposta, o engenheiro de requisitos deve iniciar a interface de captura de voz da aplicação, que obtém tudo que for dito e envia as informações para o serviço da Google de conversão de voz em texto (em inglês, Speech-To-

Text). Esta captura continua até que o entrevistador decida por encerrá-la.

O serviço da Google, que realizará a conversão dos sons em texto, tenta achar qual palavra melhor corresponde a cada som, e retorna aquela que considerar mais parecida. Esta abordagem pode gerar problemas, especialmente em estrangeirismos, gírias, contrações ou palavras que forem ditas de maneira rápida e sem clareza. Outros desafios são as situações em que o entrevistado adiciona mais ideias e informações que vêm à mente durante a fala, ou mesmo quando realiza pausas e retoma posteriormente. Tudo o que for dito é enviado ao serviço da Google, sem passar por filtragem alguma da solução desenvolvida.

Caso alguma destas situações-risco ocorra, o texto que for recebido após esta análise remota pode parecer desconexo, com palavras incorretas e sem pontuações. Para contornar estas imprecisões, caso o engenheiro de requisitos note que houve falhas, ele terá a opção de modificar esta *string*. Após feita essa validação, a solução armazena a frase da resposta em sua própria base de dados local, e então fornece uma próxima pergunta ao entrevistador.

Concluídas as perguntas sobre os atores, abrem-se outras baterias de questões genéricas e pré-definidas para cada uma dessas pessoas identificadas. Nestas, o fluxo que se segue é: para aquele dado papel, primeiro se questionará qual a ação que este deseja realizar sobre o sistema em concepção. Em seguida, para esta ação, se perguntará quais os resultados que se deseja obter ao executá-la.

Para gerar as histórias, o Speech-To-Story vai preencher um *template* de histórias de usuário, usando como parâmetro as respostas obtidas nas supracitadas perguntas que buscam atores, ações e seus resultados. De posse delas, preenche-se especificamente cada lacuna relacionada dentro do *template*, conforme Figura 5.

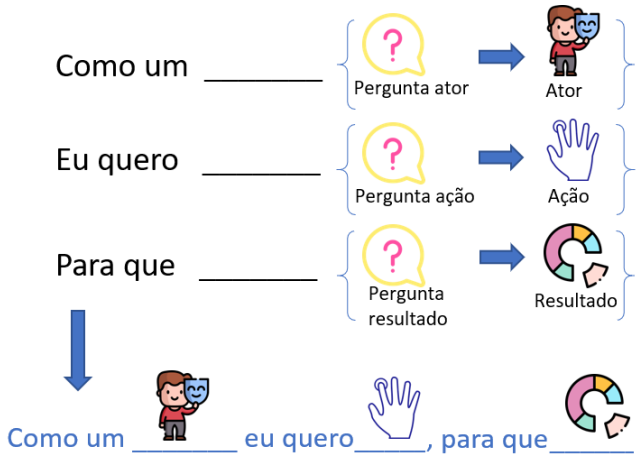


Figura 5: Template de História de Usuário - Preenchimento no Speech-To-Story.

*1) Perguntas:* Com o intuito de preencher o *template* de histórias de usuário a partir das respostas de uma entrevista, faz-se necessário formular perguntas que induzam retornos possíveis de se aproveitar para cada espaço daquele modelo. O objetivo é que elas suscitem respostas simples e curtas, contendo poucos elementos. Porém, mesmo que o entrevistado responda frases longas, a solução deve ser capaz de aceitar este retorno.

As perguntas serão oferecidas ao entrevistador, uma de cada vez, de modo que ele não pode acumular indagações. A ideia é que o sistema só apresente a próxima pergunta a ele quando se tenha recebido resposta para a questão anterior. Isso justifica porque é a partir da análise do conteúdo dessa resposta que o Speech-To-Story vai decidir qual deve ser a pergunta seguinte. Esse fluxo de questionamentos, respostas e criações de novas perguntas é observado na Figura 6.

Assim, tratando especificamente do que diz respeita à geração de histórias, a sequência de perguntas ocorre da seguinte maneira: inicia-se com as questões que visam identificar quais os papéis se relacionam com o sistema em concepção. Após captura da resposta ditada pelo cliente, o Speech-To-Story a armazena em base e sugere uma nova pergunta, que vai questionar se existem outros possíveis atores. Em caso afirmativo, mais um questionamento é apresentado, para descobrir quem seria esta pessoa. Este fluxo se repete até que não sejam identificados mais atores.

Concluídas as perguntas sobre os atores, abrem-se outras baterias de questões pré-definidas para cada uma dessas pessoas que tenham sido identificadas, seguindo o *template* descrito na Figura 7. Esta imagem descreve: para cada papel, primeiro se questionará qual a ação que este deseja realizar sobre o sistema em concepção. Em seguida, para esta ação, se perguntará quais os resultados que se deseja obter ao executá-la.

Similar ao fluxo supracitado nas questões relacionadas aos atores, a bateria relacionadas às ações também se repetirá em um *loop*, visando descobrir se existem mais elementos além daquele identificado. Esta laço continuará até que o próprio cliente afirme que não existe mais nenhuma interação que aquele usuário deva executar no sistema. Somente quando todas as ações de todos os atores forem elucidados, será liberado a criação das histórias.

Conclui-se assim que existem dois tipos de pergunta no sistema, com funções distintas: o primeiro trata de identificar um elemento único, como atores, ações e resultados, chamadas "questões identificadoras". O segundo tipo se trata das "questões de *loop*", responsáveis por manter um laço ao inquirir se há mais elementos a se considerar, quem seriam eles, e novamente se há mais outro elemento a se considerar.

Uma vez que todas as perguntas tenham sido finalizadas, com todas as respostas já captadas e armazenadas em base, o resultado é a organização dos elementos obtidos em uma estrutura similar a uma árvore, conforme se observa na Figura 8. Ela descreve que, para cada ator que tenha sido identificado, este obrigatoriamente deve ter ao menos uma ação, além de um ou mais resultados para essa interação.

A quantidade de ações possíveis é ilimitada, estando à critério do cliente dizer que já se foram identificadas todas as manipulações possíveis daquele ator. Essa mesma lógica ocorre em relação aos resultados, com a quantidade de pos-

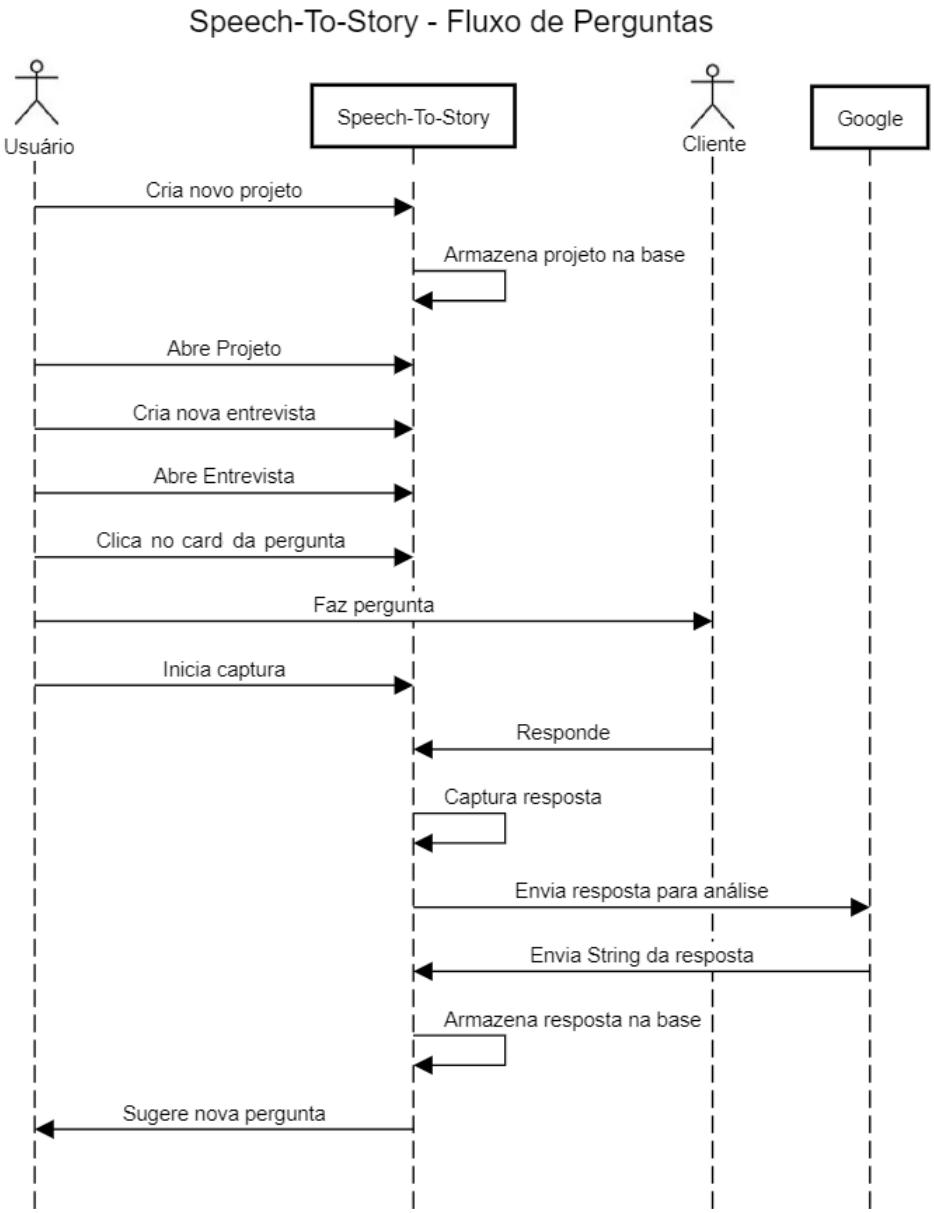


Figura 6: Diagrama de sequência da criação de uma entrevista.

síveis saídas daquela ação também não tendo limite máximo, graças à abordagem em *loop* supracitada. Observa-se, porém, que uma ação de um ator A não pode ser aproveitada para outro ator B, mesmo que seja o mesmo texto. Esta lógica também se aplica para os resultados, que são específicos de uma única ação.

O Speech-To-Story também tem capacidade de atuar em situações de respostas compostas, ou seja, em que o entrevistado já elenca diversas ações ou diversos resultados de uma única vez. É realizada uma análise para identificar se, naquilo que foi dito pelo cliente, há mais de um elemento aproveitável para o *template*. Para isso, busca-se pela quantidade de verbos e por conjunções aditivas, como "e", "também"e "bem como". Uma vez identificados, todos esses elementos podem ser armazenados já naquele momento, não atrapalhando no andamento da entrevista.

*2) Geração de histórias de usuário:* Após o usuário finalizar o fluxo de questões e respostas para todos os possíveis atores da entrevista, o Speech-To-Story libera a funcionalidade de geração de histórias de usuário. O objetivo desta função é utilizar as respostas de cada pergunta como matéria-prima para preencher *templates* de história de usuário, para então apresentá-los em formato de lista e possibilitar sua validação.

O objetivo com este preenchimento, porém, é que o Speech-To-Story seja capaz de produzir frases que estejam morfologicamente corretas. Isso porque aquela história será manipulada por outras pessoas ao longo do desenvolvimento e, portanto, deve fazer sentido ao ser lida. Para este objetivo, será utilizada

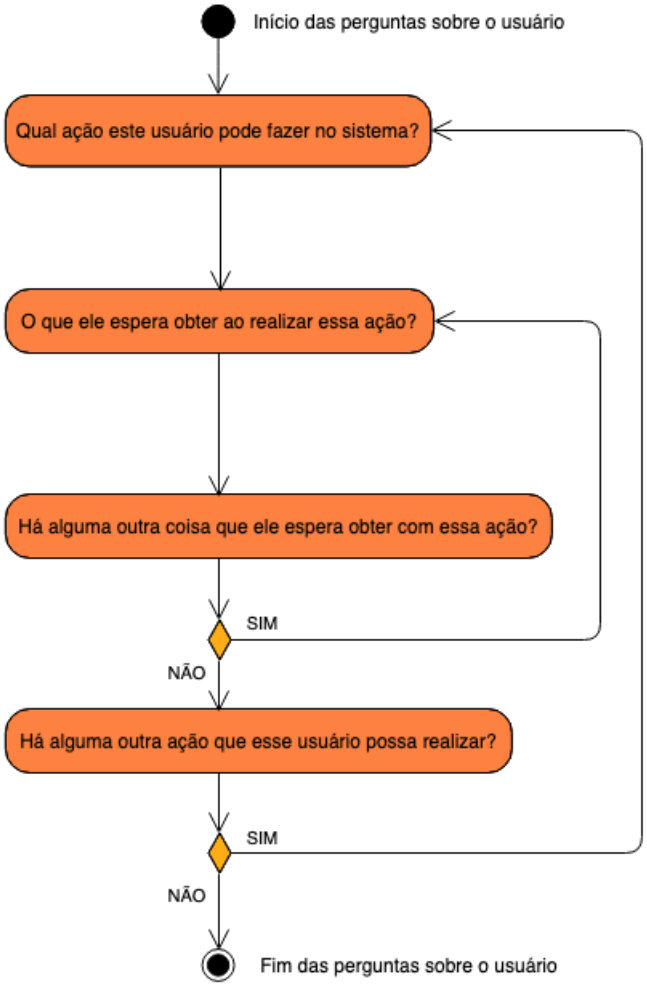


Figura 7: Sequência de perguntas para identificação das ações e resultados.

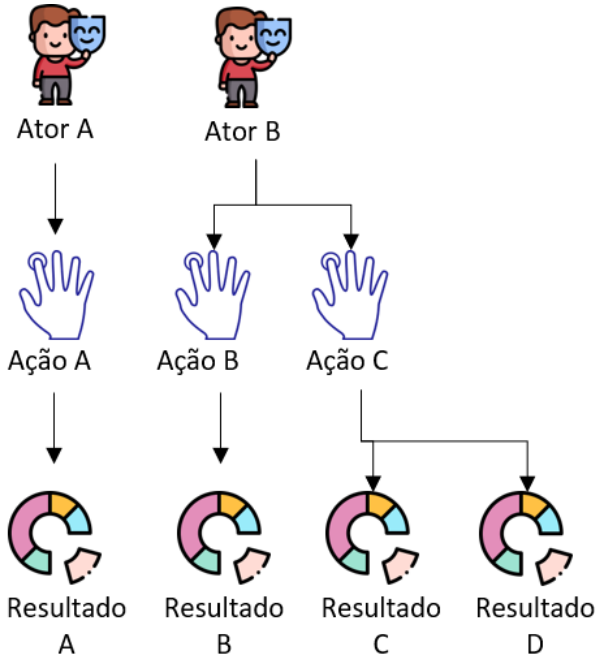


Figura 8: Visão da cardinalidade de Atores x Ações x Resultados.

a biblioteca OpenNLP em tempo de montagem das histórias, analisando as respostas de cada pergunta e avaliando quais das suas palavras, por classe morfológica, são desejadas para encaixar em cada espaço do *template*.

O processo de geração de histórias se inicia quando o usuário, dentro da *interface* da entrevista, clica no botão inferior direito, que leva à apresentação das histórias. Caso esta documentação já tenha sido criada previamente naquela entrevista, então ela será apenas exibida, em formato de lista. Caso contrário, o Speech-To-Story entende que deve-se criar histórias, inicia este processo, e congela o controle do usuário sobre a aplicação. Neste momento, uma *interface* indicando o *loading* é apresentada, para que o usuário saiba que há uma atividade em progresso, e seguirá assim até a conclusão do processo de geração das histórias.

*3) Processo de geração das histórias:* Enquanto a tela de progressão está em vigor, o Speech-To-Story inicia em *background* as ações para a criação das histórias de usuário e sua posterior apresentação. O primeiro passo é realizar a avaliação morfológica das respostas coletadas através de um *controller*, conforme trecho apresentado no Código 2. Para implementar os serviços de processamento de linguagem natural do OpenNLP, classes desta biblioteca são usadas, juntamente com um pacote binário da Apache que é específico para a análise em português.

O *controller* utiliza o binário como parâmetro para treinar a inteligência de um *Tagger* da própria biblioteca, de modo a prepará-lo para atuar sobre as respostas. A ideia é que este *Tagger* avalie cada palavra e determine qual é sua classe morfológica. Para tanto, com base no treinamento recebido, o avaliador determina qual a probabilidade de que aquela palavra pertença a cada classe. Aquela que tiver maior porcentagem de estar correta, segundo sua própria avaliação, será o valor retornado.

É necessário, portanto, separar cada palavra que constar em cada resposta do cliente, de modo a alimentar o *Tagger* e realizar esta análise. Um *Tokenizer* é quem faz esse papel, atuando sobre todas as respostas que foram salvas ao longo da entrevista, e imediatamente requisitando os serviços do avaliador. Como resultado deste *Tagger*, obtém-se dois conjuntos em *array*: um contendo as palavras que foram separadas, e outro contendo as classes morfológicas associadas às palavras do primeiro vetor.

Os dois conjuntos são enviados para a montagem de histórias em uma nova estrutura, específica para criar esta documentação. Tendo como base se aquela resposta visava identificar ator, ação ou resultado, o *controller* os redireciona para preencher o primeiro, segundo ou terceiro espaço do *template*, respectivamente. Cada preenchimento é um método distinto, que buscará por características específicas.

```
var assembly = Assembly.GetExecutingAssembly();
var resourceName = "TCC_1.ptBR-pos-maxent.bin";
Stream stream = assembly
      .GetManifestResourceStream(resourceName);

POSModel model = new POSModel(stream);
POSTaggerME tagger = new POSTaggerME(model);
WhitespaceTokenizer tokenizer =
    WhitespaceTokenizer.Instance;

string[] tokens;
string[] tags;

//Tokenize and get tags for "Como um" spot
tokens = tokenizer.Tokenize(actorAnswer);
tags = tagger.Tag(tokens);
userStory.fillFirstSpot(tags, tokens);

//Tokenize and get tags for "Eu quero" spot
tokenizer = WhitespaceTokenizer.Instance;
tokens = tokenizer.Tokenize(actionAnswer);
tags = tagger.Tag(tokens);
userStory.fillSecondSpot(tags, tokens);

//Tokenize and get tags for "Para que" spot
tokenizer = WhitespaceTokenizer.Instance;
tokens = tokenizer.Tokenize(resultAnswer);
tags = tagger.Tag(tokens);
userStory.fillThirdSpot(tags, tokens);

//Generate Story
userStory.generateStories(projectName,
                          interviewName,
                          storyModel.Id);
```

Código 2: Avaliação morfológica e chamada para gerar histórias.

Caso a resposta tenha sido para identificar um ator, então este *controller* de criação de histórias navega pelo conjunto de classes morfológicas em busca do primeiro substantivo ou pronome pessoal que encontrar, uma vez que o objetivo é descobrir quem é aquele a manipular o sistema. Esta busca foi configurada para não diferir se aquele substantivo é próprio, singular ou plural. Todo elemento que cumprir este requisito deve, portanto, preencher a primeira lacuna no *template* da história de usuário. Há, porém, uma verificação para substantivos compostos ou adjetivados: o *controller* avalia se a próxima palavra também é um substantivo ou adjetivo, e acumula à primeira palavra encontrada, de modo a adicionar a combinação ao *template*. Caso não se configure um substantivo composto, então apenas aquele primeiro substantivo é adicionado à história.

Para a segunda lacuna da história, que visa identificar qual ação foi realizada pelo ator, o *controller* busca quais palavras foram classificadas como verbo. São aceitos verbos na forma base, no passado, gerúndio, presente-particípio, passado-particípio e mesmo em terceira pessoa do singular no presente. Há, porém, situações em que o OpenNLP avalia erroneamente alguns verbos como sendo substantivos. Em face a isso, foi implementada uma estrutura de *double-check*, usando o mesmo *Tagger*, que avalia qual a segunda classe gramatical mais provável para aquela palavra. Se for verbo, ela também deve ser considerada para a história. Uma vez identificado o verbo, o sistema passa a acumular todas as palavras que vierem a seguir na resposta, sem qualquer filtragem ou busca por classes específicas.

O motivo da não-necessidade dessa busca se explica pois a própria pergunta original já induzia uma resposta que indique ação, de modo que entende-se que o que vier após o verbo seria um complemento a esta. Para além disso, caso naquela resposta tenha sido identificada mais de uma possível ação pela própria aplicação em tempo de entrevista, entende-se que o engenheiro de requisitos já recebeu uma *interface* para analisar se são de fato ações diferentes ou se tratam do mesmo ato. A frase sendo analisada no momento é, portanto, uma decisão do próprio engenheiro identificando que ela toda se trata de uma única ação. Desta forma, o sistema se exime de realizar uma nova análise, pois compreende-se que ela já foi feita anteriormente pelo usuário.

Com o intuito de preencher o terceiro espaço da história, de saídas esperadas, o sistema volta a buscar por palavras que tenham sido avaliadas como verbos, mas também por substantivos. Como esta terceira lacuna visa preencher o trecho "para que eu", entende-se que o resultado, qualquer que seja ele, será iniciando a partir de um verbo, a exemplo de "obter", "ver"ou "gerar". Todavia, para acomodar possíveis situações em que não existam verbos, o Speech-To-Story passa a preencher o *template* a partir do primeiro substantivo que encontrar, como se o entrevistado falasse apenas do objeto que será resultante da ação. Exemplos disso são "listas", "relatórios"e afins. Nestes casos, o próprio sistema adiciona na história o verbo "obter", para melhor compreensão humana, a exemplo de "obter relatórios de consumo".

Uma vez que as três lacunas do *template* estejam preenchidas, um modelo de história é criado com o texto resultante deste preenchimento. Ele é identificado através de um ID único, permitindo posteriores clones e edições, além de receber informações sobre o entrevistado e o projeto ao qual pertence. Finalmente, agora que está concluída, essa história é salva no banco de dados.

Em uma visão mais alto nível, o fluxo final de geração das histórias, desde a solicitação até a exportação do documento, pode ser observado na Figura 9.

*4) Apresentação das histórias:* Uma vez finalizada a criação de histórias e seu armazenamento em banco, a *interface* de histórias de usuário é carregada. Todas as histórias daquela entrevista serão consultadas no banco, para que então sejam carregadas em memória e apresentadas em formato de lista, sob um campo editável. Esta *interface* é observável na Figura 10.

Neste momento, o engenheiro de requisitos deve avaliar os textos gerados. Caso note alguma inconformidade, pode-se alterar o conteúdo errôneo e salvá-lo em base. Também é possível clonar histórias, caso se deseje separá-las em duas. Finalmente, é possível apagar aquela história caso ela não seja desejada.

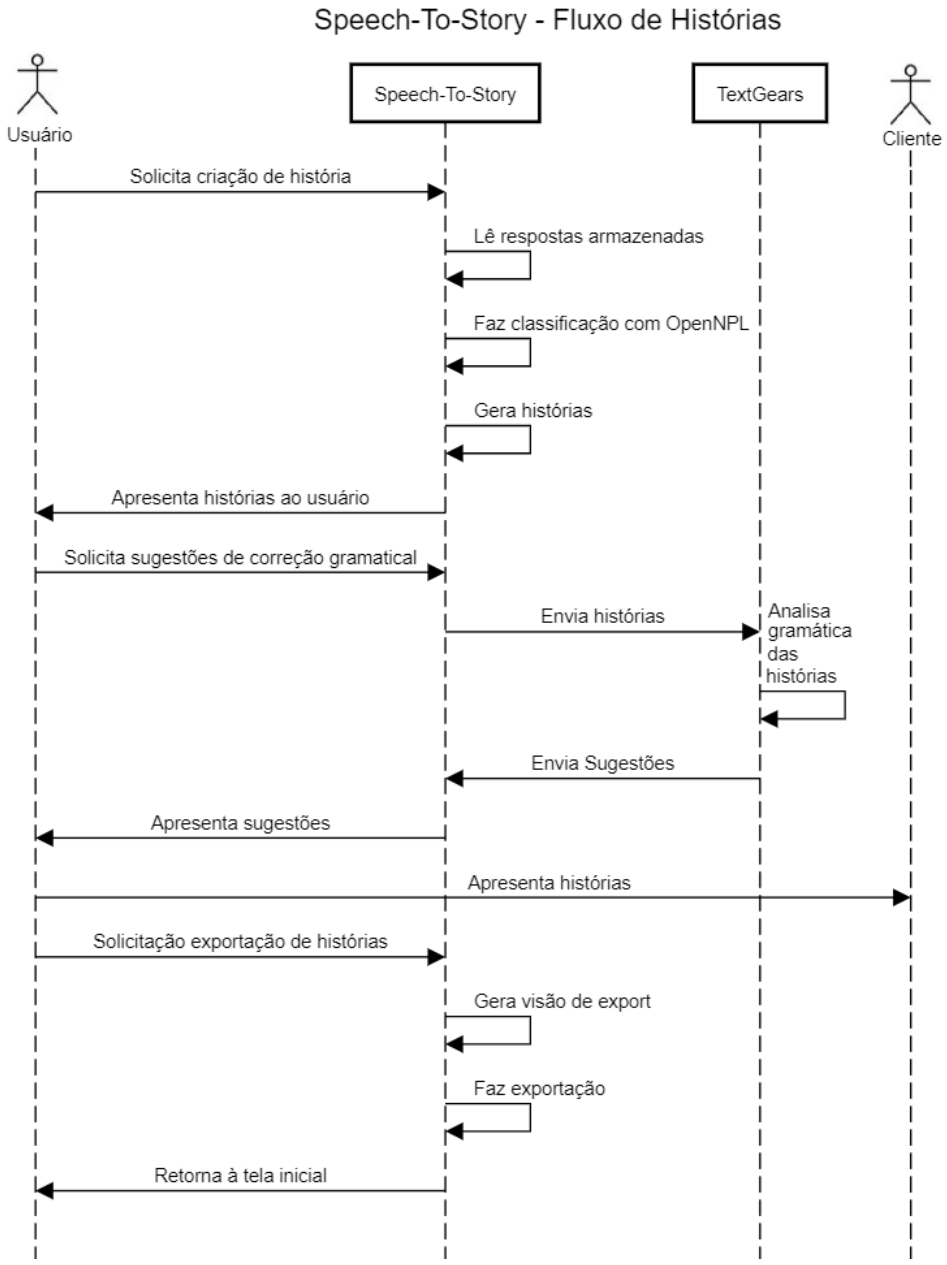


Figura 9: Diagrama de sequência da finalização da entrevista e criação das histórias de usuário.

Caso queira incluir novas histórias além das que foram geradas, o *floating button* de adição permite a criação manual destas. Por fim, no canto superior direito, é possível exportar todas as histórias, em uma visão de *card*, para que o cliente possa validá-las.

*5) Histórias relacionadas a TI:* No momento que a *interface* de histórias é carregada, o texto de cada delas é analisado para verificar se contém informações relacionadas a TI. Para isso, um conjunto de palavras comumente utilizadas na área foi definido em arquivo JSON, para ser utilizado em comparação com cada palavra que compõe o texto. Isso porque as histórias de usuário têm como objetivo auxiliar o time de desenvolvimento com exemplos de interação do usuário e do negócio, não podendo portanto descrever histórias que envolvam termos de TI. A ideia é preservar o conceito original das histórias, as quais devem ser vistas sempre do ponto de vista de alguém alheio a terminologias de sistema.

Porém, mesmo que sejam identificadas histórias relacionadas a TI, não se faz interessante descartar este documento, pois é possível que o sistema sendo elicitado seja de fato um sistema de TI, o que tornaria esta história válida. Para acomodar essa possibilidade, quando as histórias contendo termos técnicos são apresentadas, um alerta é adicionado para aquele *card*, como na Figura 11. A ideia é indicar ao engenheiro que aquela pode não ser uma história válida, apresentando uma mensagem com essa informação, quando


Figura 10: Histórias de usuário.


Figura 11: História relacionada a TI.

clicada. Fica a cargo do profissional aceitar o alerta e apagar a história, ou recusar e mantê-la.

*6) Correção gramatical:* Uma vez que as histórias estejam criadas e dispostas em formato de lista, o Speech-To-Story ainda oferece a possibilidade de realizar uma análise gramatical sobre elas. Com o intuito de corrigir possíveis falhas de concordância ou tempo verbal incorreto, um *toggle* para sugestões é disponibilizado no canto esquerdo. Ao ativá-lo, as histórias de usuário são enviadas, uma a uma, em requisições REST para o avaliador gramatical da *TextGears*. Esta funcionalidade só está disponível quando o aparelho celular possui conexão à internet.

Esta ferramenta externa analisa as histórias e, como resultado, é recebido um JSON indicando as palavras incorretas, sua posição dentro da frase, e uma sugestão de sua correção. De posse desta resposta, a solução vai, internamente, montar uma nova história com aquela sugestão recebida e apresentá-la ao usuário. O engenheiro tem a opção de aceitar a sugestão, substituindo a história original, ou recusá-la. É possível observar como uma sugestão é oferecida ao usuário na Figura 12, bem como as opções de aceite ou recusa.

### *D. Arquitetura da Solução*

De maneira a facilitar a leitura e entendimento do código desenvolvido, é possível dividir a arquitetura da solução em módulos. O objetivo de cada um deles é gerenciar o ciclo de vida de suas entidades, através de funções como criação, edição e deleção. Esta organização pode ser detalhada na Figura 13, e será explicada nas subseções que se seguem. Também evidencia-se na imagem a conexão entre os módulos, representando mudanças de contexto da aplicação. Um exemplo prático disso é quando, a partir de um projeto do tipo Manual, a aplicação leva o usuário até a entidade História, do módulo 4. A base de dados, porém, pode ser manipulada por qualquer atividade, independente de qual módulo pertença.

*1) Projetos:* O módulo de Projetos tem como responsabilidade gerenciar todo o ciclo de vida dos *models* que representam projetos de desenvolvimento de software, podendo ser dos tipos “Fala” ou “Manual”. O primeiro se trata do fluxo principal da aplicação, ao passo que o segundo é o alternativo.

O projeto de tipo “Fala” atuará como um agregador de entrevistas, contendo perguntas, respostas e posteriores histórias. Ele se difere do “Manual”, que apenas agrupa histórias de usuário criadas manualmente naquele projeto. Em comum a ambos, está a ideia de que um projeto é somente uma casca para agrupar outras entidades sob si.

Ao ser iniciado, este módulo busca no banco de dados os projetos já existentes, para apresentá-los em lista para o usuário. Ele também é o responsável por criar novos projetos e persistí-los em base. Por fim, possui uma comunicação direta com o módulo de histórias, pois esta documentação é também vinculada ao ID daquele projeto, possibilitando que se exporte as histórias de todas as suas entrevistas de uma única vez.


Figura 12: Sugestão de correções na história.

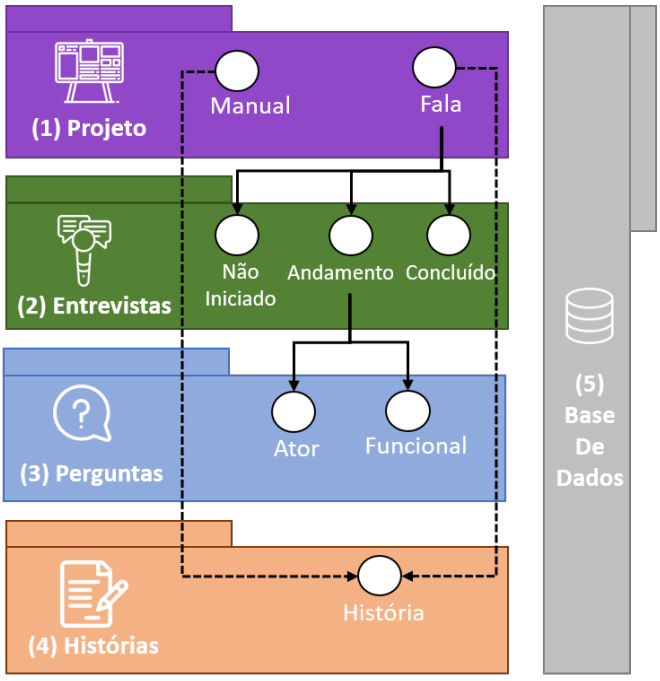


Figura 13: Arquitetura da Solução - Visão das Entidades.

*2) Entrevistas:* O módulo de Entrevistas se responsabiliza por gerenciar todo o ciclo de vida das entrevistas de um determinado projeto. Um projeto do tipo Fala, que descreve o fluxo principal da solução, é composto por diversas entrevistas. Cada uma delas representa um encontro com o entrevistado, em que perguntas são feitas, e suas respostas obtidas.

Uma entrevista recém-criada se encontra com estado "não iniciado", uma vez que nenhuma pergunta foi ainda realizada; Passa para "andamento", se já houver perguntas respondidas, porém não todas tenham sido completadas; Por fim, "concluído", quando todas as questões já foram realizadas e as histórias de usuário tenham sido criadas. Este é um módulo que atua como intermediário entre os módulos de perguntas e histórias.

De um lado, ele se comunica com o módulo de perguntas, obtendo deste se todas as questões já foram respondidas ou não, com o intuito de avaliar se já é possível gerar histórias de usuário. Do outro lado, quando as condições para criar as histórias tenham sido cumpridas, o módulo de entrevistas ativa o de histórias, inicializando as suas ações. Como atividade opcional, este módulo também é capaz de realizar funções de *back-office* para corrigir manualmente possíveis falhas em respostas obtidas no módulo de perguntas.

*3) Perguntas:* Este módulo possui como função o gerenciamento das perguntas, gerando novas questões, interagindo com a *interface* de captura de voz, enviando e recebendo requisições para os serviços *speech-to-text* da Google, armazenando suas respostas, e garantindo que a sequência das perguntas está ocorrendo da maneira planejada. Os questionamentos são modelados e divididos em dois tipos, conforme seus objetivos: as identificadoras, que visam obter elementos funcionais acerca do sistema elicitado, como seus atores, ações e resultados, e as questões de *loop*, que servem para controlar o andamento a entrevista.

Ao início de cada entrevista, este módulo começa oferecendo perguntas para identificar os atores. Para isso, ele consome um arquivo JSON, dentro do próprio Speech-To-Story, que contém todas as questões pré-formuladas. Ao obter todos os atores, inicia-se as perguntas acerca de cada um deles, para obter quais as suas interações possíveis sobre o sistema, bem como extrair quais os resultados que este deseja obter a cada interação.

Este módulo conta com uma classe de *controller* para verificar quais perguntas já foram respondidas e quais ainda estão pendentes, usando essa avaliação para definir qual deve ser a próxima pergunta a ser realizada. Para cada questionamento, é também responsabilidade deste módulo iniciar a captação da resposta do entrevistado, quando for requisitado pelo entrevistador. Faz-se a requisição ao microfone do celular, a conexão com o serviço da Google de transcrição da fala, e encerra-se essas ações quando solicitado pelo usuário.

Finalizada a captura, também é neste módulo que o engenheiro pode realizar ajustes no texto obtido na transcrição, e seu resultado é então salvo no banco de dados. Quando não há mais perguntas a realizar, o controlador encerra a sessão,

e indica ao módulo de entrevistas que suas atividades estão concluídas.

*4) Histórias:* O módulo de Histórias é responsável por gerar, apresentar e gerenciar histórias de usuário através de seus *models*. Em cada entrevista, este só pode ser ativado quando todas as perguntas acerca de todos os atores estão respondidas. Ao ser acionado, o módulo de histórias busca todas as respostas que foram salvas naquela entrevista. Para cada uma delas, é realizada uma análise morfológica usando a biblioteca OpenNLP da Apache. Caso a palavra avaliada seja da classe desejada, o *template* de histórias de usuário é preenchido, e a história é salva na base.

Ao iniciar a *interface* de apresentação, estas histórias armazenadas são lidas e expostas em formato de lista, podendo ser editadas, apagadas ou duplicadas. Este módulo ainda oferece a possibilidade de realizar uma checagem gramatical sobre as histórias, com o intuito de avaliar se respeitam regras de concordância e identificar possíveis erros gramaticais.

Por fim, este módulo também é responsável pela funcionalidade de exportação de histórias, montando uma visão gráfica de cartões de papel contendo aquelas frases, e exportando-as para outros aplicativos instalados no aparelho. São ofertadas tanto as apresentações via formato PDF, quanto em texto puro para possibilitar manipulações, via TXT. Projetos do tipo "Manual" apenas utilizam diretamente as funcionalidades do módulo de Histórias, não tratando de entrevistas nem perguntas.

*5) Base de dados:* Com o intuito de manter os dados armazenados localmente, optou-se pela utilização de um banco SQLite, dentro do próprio aplicativo. A modelagem do banco reflete os *models* que são utilizados no Speech-To-Story, com tabelas que correspondem a suas entidades, e colunas que correspondem aos seus atributos. É possível observar esta disposição e relações na Figura 14.

Ao longo do fluxo de execução do Speech-To-Story, são realizadas consultas ao banco para listar projetos, entrevistas e histórias de usuário. Também são feitas chamadas de *insert* usando *scripts* SQL sempre que se cria uma nova entidade, como projeto ou entrevista. *Updates* são realizados na tabela de perguntas, sempre que a solução vai salvar uma resposta capturada. *Deletes* são feitos quando se apaga um projeto, entrevista ou história de usuário não desejada.

Seu uso, porém, apresenta algumas desvantagens: a ausência de chaves primárias impossibilita que se realize *joins*. Foi necessário solucionar esta situação em código, no momento da consulta, efetuando buscas em cada tabela separadamente para, a seguir, comparar seus resultados através dos *models* obtidos. Neste momento, finalmente, avalia-se se há atributos em comum dentre aquelas duas ou mais entidades.

### *E. Fluxo da Solução*

A solução desenvolvida é composta de um fluxo principal, motivador deste trabalho, e um fluxo alternativo, cujo intuito é ampliar as capacidades e funcionalidades da aplicação. Ambos são apresentados nesta seção.

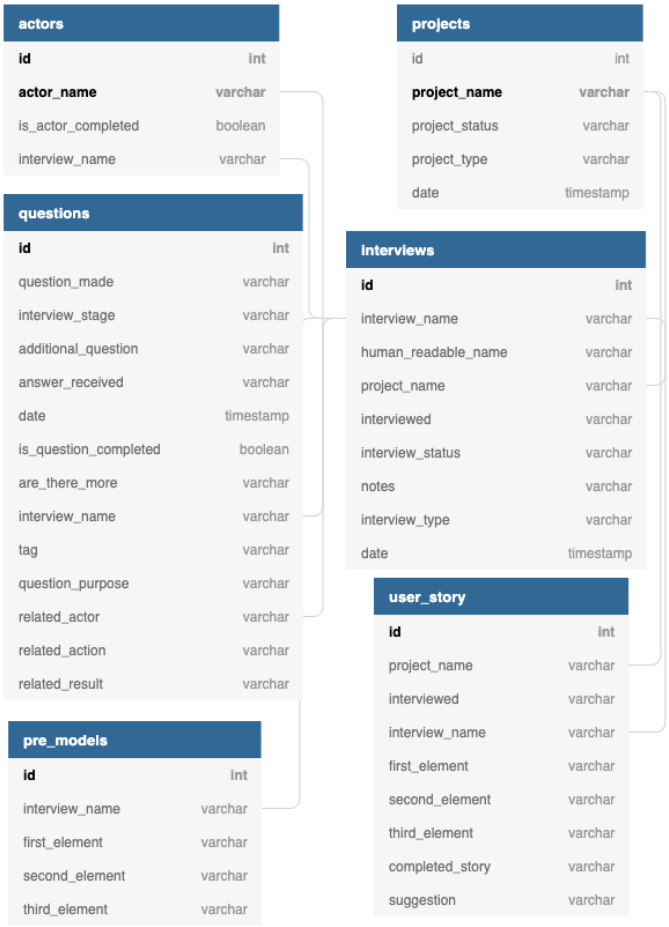


Figura 14: Modelo do Banco de Dados.

Ao inicializar o Speech-To-Story, o sistema busca em base quais projetos já existem armazenados. É obtida uma lista, expondo *cards* dispostos verticalmente, cada um representando um projeto e contendo suas informações como nome e data. Para efeitos de organização, separa-se em abas distintas os projetos de tipo "Fala" e "Manual".

De modo a seguir com este fluxo de criação de histórias, será necessário utilizar um projeto de "Fala" já existente ou criar um novo. Para a segunda opção, apresenta-se a interface de criação de projetos como na Figura 15, solicitando do usuário apenas o nome que se deseja atribuir. Clicando no campo de texto, o próprio Android inicia o teclado para que se digite. Opcionalmente, no topo, há um *toggle* que marca se o projeto é do tipo manual ou não. Neste fluxo, esta chave estará desativada.

Ao confirmar, no botão do canto inferior direito, um novo projeto é criado e o usuário retorna para a lista de projetos. O seu último elemento será um novo cartão, representando esta entidade criada, contendo o nome escolhido, um ID único que é gerado, data e seu tipo, "Manual" ou "Fala". Caso exista um projeto de mesmo nome, isto é tratado, para evitar conflitos, através de um numeral adicionado ao final do título. Neste ponto, ao clicar no cartão correspondente àquele projeto, a

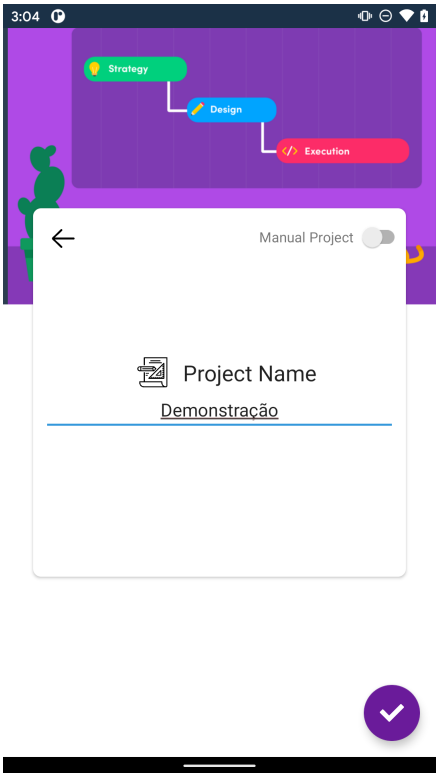


Figura 15: Criação de novo projeto.

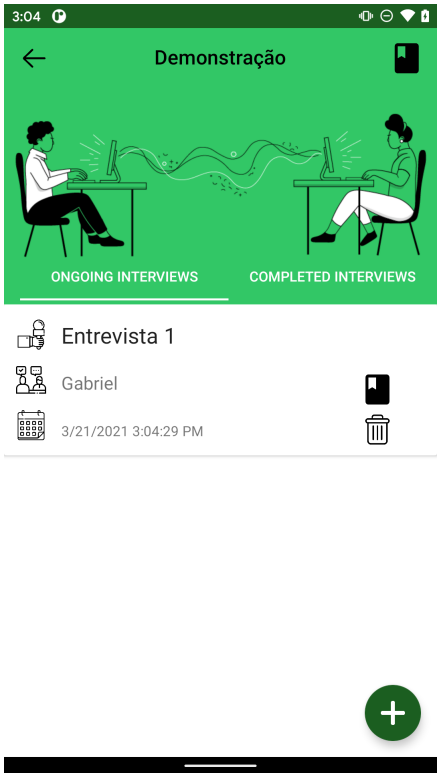


Figura 16: Nova entrevista apresentada.

sua lista de entrevistas será apresentada, estando inicialmente vazia.

Faz-se necessário, portanto criar novas entrevistas. Ao clicar no botão no canto inferior direito, inicia-se a *interface* de criação de novas entrevistas, onde é possível nomear aquela conferência, definir quem é o entrevistado, e adicionar possíveis observações nas notas. Ao confirmar, retorna-se automaticamente para lista de entrevistas, em que aquela recém-criada estará presente, conforme evidenciado na Figura 16. Ela apresentará atributos refletindo as informações definidas pelo usuário durante sua criação, além de outras complementares geradas pelo próprio sistema, como data e hora de criação, ID único e identificação do projeto ao qual pertence.

Ao clicar naquela entrevista, a sua *interface* é apresentada. O engenheiro de requisitos deve proceder para a realização de perguntas, com o intuito de avançar na progressão desta entrevista. Neste exemplo, como a entrevista acabou de ser criada, deve-se primeiro buscar identificar quem são os atores que vão manipular o sistema a ser desenvolvido. Para isso, deve-se pressionar o botão que indica esta ação.

Este clique vai iniciar a *interface* de perguntas, onde se apresenta ao usuário as questões que devem ser feitas ao entrevistado. Questionamentos e respostas são dispostos em uma lista vertical mas, como este momento se trata apenas da busca por atores, então apenas os pares relativos a este objetivo serão apresentados. Neste caso, como a entrevista foi recém-iniciada, há apenas a pergunta, ainda não respondida: "quem é a principal pessoa que utilizará este sistema?". Ao clicar no *card* daquela pergunta, uma nova *interface* é iniciada, conforme a Figura 17.

Nesta, será apresentada aquela pergunta que se deseja fazer ao entrevistado, na parte superior da estrutura. Há ainda instruções, na parte inferior, indicando que para iniciar a captura da resposta, deve-se clicar no microfone. Neste ponto, o entrevistador deve realizar a pergunta descrita em voz alta. Só aí, quando o entrevistado for responder, o botão do microfone deve ser pressionado.

Ao clicá-lo, o Speech-To-Story começa a a gravar a resposta do cliente. Essa ação necessita de acesso à internet, bem como da permissão de acesso ao microfone do celular. Caso esta não esteja concedida, o próprio Android inicia o *pop-up* solicitando seu uso. De posse da permissão, o aplicação passa a captar os *inputs* de voz obtidos pelo microfone do aparelho. Quando entender que o entrevistado finalizou a resposta, o entrevistador clica novamente no botão do microfone, interrompendo a captura de voz e já enviando para transcrição.

A resposta ditada pelo cliente e formatada em texto é apresentada ao usuário, de modo a possibilitar edições manuais para corrigir eventuais falhas na transcrição. Ao confirmar que aquele texto está pronto e correto, ele então é salvo em base, sem passar por nenhum outro tratamento. Caso a captura falhe, seja por problemas no microfone ou no envio ao servidor da Google, a *interface* de perguntas será novamente apresentada, e o conteúdo falho é descartado. Desta forma, pode-se repetir a questão e o fluxo não será comprometido.

Alternativamente, caso por fatores externos não seja pos-


Figura 17: *Interface* de uma pergunta.

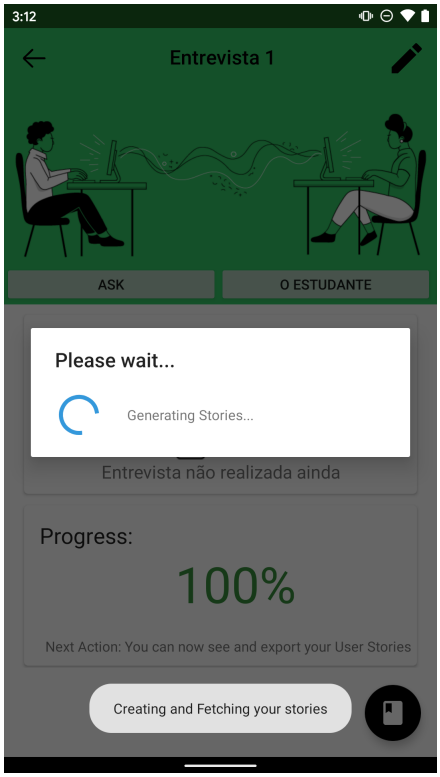


Figura 18: *Loading* enquanto Speech-To-Story gera histórias.

sível realizar a gravação da resposta naquele momento, há uma opção na parte inferior ainda na Figura 17 para que o engenheiro de requisitos digite a resposta que receber. Desta forma, o profissional pode dar sequência na entrevista mesmo quando não houver condições sonoras naquele momento.

Seguindo com o fluxo principal, como uma pergunta identificando atores já foi respondida, o sistema adiciona à lista a próxima a ser feita: uma de *loop*, com o intuito de questionar se existe mais algum outro papel ou não. Caso responda-se que não, então a solução entende que todos os atores foram obtidos, e nenhuma nova questão será criada, finalizando esta sessão.

Após identificados os atores, independente de quantos sejam, passa-se à identificação das ações de cada um deles sobre o sistema, bem como os resultados destas manipulações. Para isso, de volta à *interface* de entrevista, um botão será adicionado para cada papel obtido. Ao clicar em um destes, será apresentada uma nova pergunta específica daquela pessoa.

O fluxo nesta sessão é similar ao realizado anteriormente para buscar os atores: identifica-se qual a ação, e posteriormente para aquela ação, qual o resultado esperado ao realizá-la. Uma vez obtida esta saída, é verificado se existe mais alguma outra para aquela mesma ação. Se houver, pergunta-se qual é. Caso contrário, em não havendo outro efeito resultante daquela interação com o sistema, então a solução fornece a pergunta que questiona se existe alguma outra ação realizada por aquela pessoa. Em caso positivo, o fluxo se repete. Em negativo, nenhuma pergunta será mais criada, e entende-se que já foi coletado tudo que se deseja acerca daquele ator.

Uma vez finalizada a sessão, cada resultado único obtido será atrelado à sua ação, e esta ação ao ator em questão. Desta forma, cada ator pode ter mais de uma ação, e cada ação pode gerar mais de uma saída.

Uma vez que sejam respondidas todas as ações e resultados de cada um dos papéis, portanto não restando nenhuma pergunta pendente, a *interface* da entrevista exibe o estado de concluída. Neste momento, o botão da parte inferior direita é liberado, e através dele, pode-se criar histórias.

Ao clicar neste botão, o Speech-To-Story inicia o processamento para criação desta documentação, e a *interface* da aplicação apresentará um *loading* enquanto isso ocorre, como na Figura 18.

Quando as histórias são finalmente geradas, então a *interface* de apresentação das histórias de usuário é carregada. Elas são exibidas em formato de lista, porém sob um campo editável. Apenas as histórias daquela entrevista específica estarão disponíveis.

Todavia, como um projeto é composto por diversas entrevistas, é possível pela *interface* do projeto, solicitar a apresentação de todas as histórias existentes, de todas as entrevistas que existem sob aquele determinado projeto. Há opções para manipular cada uma delas, à sua direita: é possível clonar, salvar as alterações feitas no campo editável, e finalmente apagá-las. Com o intuito de corrigir possíveis falhas de concordância ou de tempo verbal, um *toggle* para obter sugestões é disponibilizado.

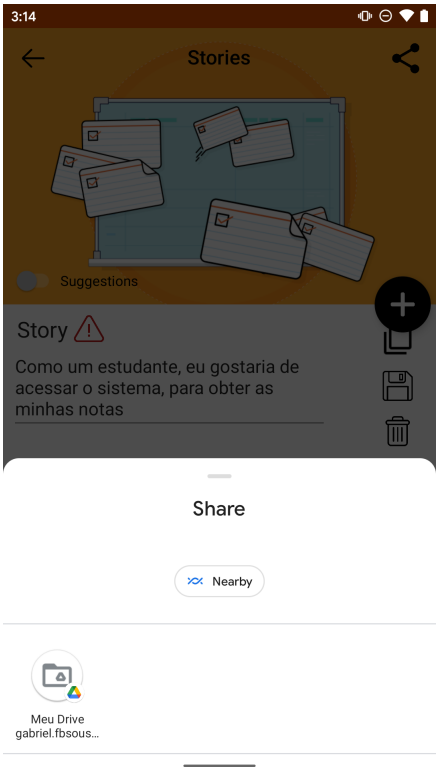


Figura 19: Exportação de histórias.

Finalmente, há no canto superior direito a opção de compartilhamento das histórias, abrindo uma *interface* como a da Figura 19. Uma vez solicitada esta operação, abrem-se duas opções: é possível exportar aquelas histórias em formato de texto, por TXT, ou optar por exportá-las em uma formatação visual, por PDF, simulando cartões reais contendo também o nome do projeto e o nome do entrevistado.

Independente da exportação escolhida, a aplicação solicita ao Android o uso da sua *interface* de compartilhamento. A partir deste ponto, todas as atividades são realizadas pelo próprio sistema operacional, conforme Figura 20. Quaisquer aplicações que aceitem o formato escolhido estão aptas para receber um arquivo com as histórias exportadas. Desta maneira, encerra-se o fluxo principal do Speech-To-Story.

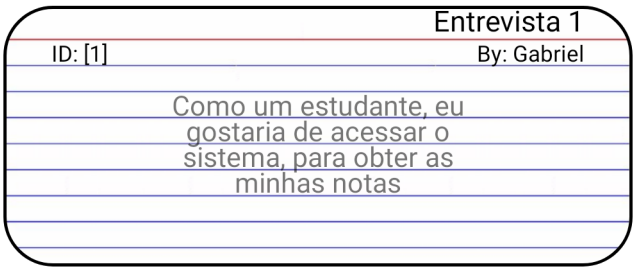


Figura 20: História exportada via fluxo principal.

*1) Fluxo Alternativo:* Enquanto uma ferramenta CASE, o Speech-To-Story pode também dar suporte para engenheiros de requisitos em outros momentos além da entrevista. Desta forma, foi desenvolvida uma funcionalidade que permita a geração de histórias de modo manual, transformando a solução em um gerenciador de histórias de usuário, administrando seu ciclo de vida diretamente.

Para chegar a este resultado, o usuário apenas necessita criar ou abrir um projeto do tipo "Manual", ao invés daqueles de tipo "Fala". Eles são mais limitados, pois não lidam com perguntas, respostas e captação de voz. Seu intuito é apenas criar histórias de usuário manualmente.

Iniciando o fluxo, cria-se um novo projeto, porém ativando o *toggle* para indicar que deseja-se um projeto manual. Dá-se um nome à nova entidade, confirma-se a operação, e retorna-se para a lista de projetos.

Ao clicar sobre o projeto criado, o usuário é levado diretamente à *interface* de histórias de usuário, que estará inicialmente vazia. Por se tratar de um projeto manual, a única maneira de preencher esta lista é clicar no botão de adição, que inicia uma nova *interface* representada na Figura 21.

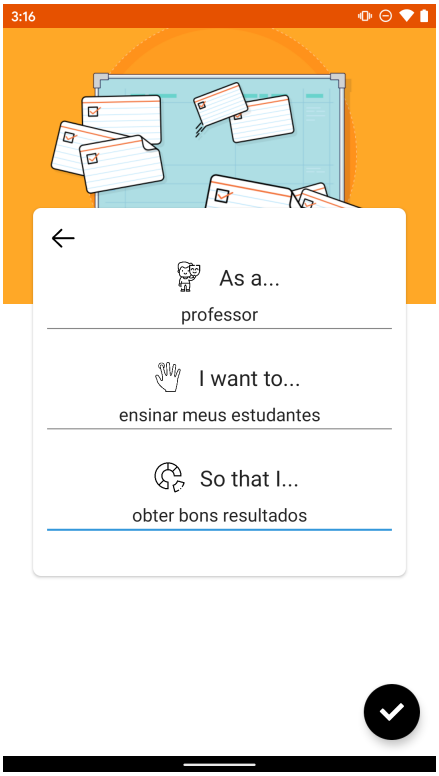


Figura 21: Criação de nova história manualmente.

Neste momento, o engenheiro deve preencher os campos de texto, com as informações que a história deve conter, completando assim o *template*. Ao clicar no botão de finalizar, o mesmo processamento de análise morfológica descrito no fluxo principal, utilizando a biblioteca OpenNLP, também é executado sobre o texto que foi escrito. A história então é salva e, automaticamente, retorna-se para a lista de histórias,

onde o novo elemento estará também listado.

Todas as opções de manipulação detalhadas no fluxo principal também são aplicáveis e funcionais. Similarmente, conta-se com alertas para possíveis histórias relativas a TI e sugestões através de requisições ao *TextGears*. Não há diferenças na exportação, em relação aos projetos de tipo "Fala": as histórias manuais também podem ser compartilhadas em formatos de texto ou PDF, resultando em cartões como evidenciado na Figura 22.

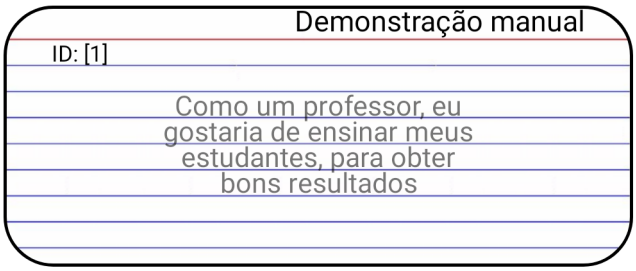


Figura 22: História exportada via fluxo alternativo.

### F. Distribuição

O Speech-To-Story ainda não está disponível para uso público. Entretanto, para executar alguns testes, pode ser feita instalação da aplicação na *Play Store* da Google através do link: https://play.google.com/store/apps/details?id=com.gabriel.speech_to_story. O acesso está disponível apenas para o usuário da Google SpeechToStory.Tester@gmail.com e da senha Speech123. A ideia é futuramente disponibilizar a aplicação para uso gratuito, porém o código do software será fechado. Além disso, um vídeo com uma demonstração de uso da ferramenta pode ser visualizado no endereço https://youtu.be/8rYpD3YPuEU.

## V. Trabalhos correlatos

### A. Requirements Apprentice

De forma similar à solução proposta, o Requirements Apprentice (RA) [37] se trata de uma ferramenta automatizada para auxiliar no processo de levantamento de requisitos. O foco do RA está nas fases iniciais do processo de elicitação, visando atacar problemas de ambiguidade, contradição e incompletude. Esta ferramenta se propõe a auxiliar o analista no desenvolvimento dos requisitos de um *software*.

Para realizar esta atividade, o RA armazena as informações advindas do *input* do usuário em uma base de conhecimento chamada *Requirements Knowledge Base* (RKB). Através de bibliotecas chamadas *cliché*, ele é capaz de categorizar termos em domínios específicos, tornando esta ferramenta aplicável a diversos negócios. Mais *clichés* podem ser adicionados, fazendo com que o RA possa dar suporte a novos domínios.

O objetivo do RA difere da solução proposta, uma vez que seu intuito é de ajudar o analista a lembrar de detalhes esquecidos do projeto e corrigir erros. Outra diferença é que, ao final dos processos daquela ferramenta, a informação guardada no RKB pode ser transformada em um documento de requisitos tradicional, ao invés de histórias de usuário. Por fim, sua plataforma é *desktop*, enquanto a solução apresentada é *mobile*. O processamento de linguagem natural no RA é utilizado é para categorização em grupos, e não para auxílio na montagem de requisitos. Finalmente, não é realizada nenhuma análise gramatical sobre a saída.

Apesar dessas diferenças, há correlação com o Speech-To-Story no recebimento de *inputs* por voz, a partir do cliente, além de ser uma aplicação pensada igualmente para auxiliar o engenheiro de requisitos durante a entrevista. Compartilha-se também o objetivo de criar automaticamente a documentação, a partir dos dados coletados em entrevistas, embora em formatos diferentes.

### B. Knowledge-Based Requirements Assistant

O Knowledge-Based Requirements Assistant (KBRA) [38] é parte de um sistema maior, chamado Knowledge-Based Software Assistant: uma proposta que visa aplicar conceitos de inteligência artificial aos processos de projeto e implementação de sistemas. O objetivo do KBRA é realizar a formalização de requisitos para auxiliar no desenvolvimento de *software*.

Esta também se trata de uma ferramenta assistiva, provendo diversos métodos para que o analista possa descrever e apresentar requisitos. O sistema constrói uma base de conhecimento em cima dos requisitos coletados, permitindo que se realize leitura e atualização das exigências coletadas.

O KBRA classifica todos os requisitos armazenados em grupos. Com base nisso, ele provê um *framework* dedutivo para encontrar contradições nos requisitos. Uma vez que encontre, esta ferramenta provê suporte para retração, explanação e identificação dos culpados.

A principal diferença do KBRA para o sistema proposto é o fato de que ele se trata de um sistema para gerenciamento de requisitos. Além disso, a entrada dos dados é manual, e não se gera documentação.

Por outro lado, há similaridades: o KBRA também é uma ferramenta CASE para apoio ao processo de levantamento de requisitos. A semelhança mais relevante, porém, é que este também implementa uma camada de processamento de linguagem natural, embora com o objetivo de avaliar um requisito sobre o outro.

O KBRA, ao final, não resulta em documentação alguma. O processamento de linguagem natural utilizado é para avaliar requisitos correlatos, e não para apoiar na montagem destes requisitos. Também não é realizada nenhuma correção gramatical.

### C. FAES

Similarmente à solução proposta, o FAES [20] é uma ferramenta CASE cujo intuito é auxiliar na realização de entrevistas. Para isso, esta aplicação provê conjuntos de questões para guiar o engenheiro de requisitos durante a conferência, analisando as respostas e propondo novas questões. Ele também armazena informações em uma base de conhecimentos,

a qual guarda observações que o entrevistador deseje fazer, e que pode ser acessada através de relatórios.

À semelhança da solução proposta, o FAES atua na atividade de entrevistas, do processo de levantamento de requisitos. Ao prover questões, ele ajuda a tornar a entrevista no tipo estruturada, tentando diminuir problemas como ambiguidade e requisitos incompletos. Também similar ao Speech-To-Story, o FAES oferece uma solução automatizada, gerando novas perguntas para obter, enquanto a entrevista acontece, novas informações sobre o sistema.

Diferentemente da solução proposta, o intuito do FAES não é gerar documentação. O *input* das respostas precisa ser feito manualmente, ao passo que o Speech-To-Story prioriza este processo através da captação de voz. Esta é uma aplicação *desktop*, necessitando sempre que o engenheiro de requisitos esteja com um *notebook* no momento da entrevista.

O FAES, por fim, não gera documentação alguma. Seu processamento de linguagem natural tem como objetivo avaliar as respostas apresentadas e oferecer novas perguntas ao usuário, recurso que também está presente no Speech-To-Story, embora não utilizando linguagem natural. Todavia, o FAES não gera documentação.

### *D. JIRA Agile*

Esta ferramenta provê uma solução completa para dar suporte ao gerenciamento de projetos, com foco em metodologias ágeis [39]. Uma das funções disponíveis é a de criação de histórias de usuário, semelhante à solução proposta. Com as histórias criadas, ele provê alternativas para sua manipulação, como visualizar as atividades que o time realiza sobre elas e gerar relatórios com o progresso das tarefas.

Outras similaridades desta ferramenta com a solução proposta são a possibilidade de uso em ambientes *mobile*, além dos objetivos de documentar as histórias de usuário a partir de entradas do engenheiro de requisitos, bem como gerir de ciclo da vida da história posteriormente.

Por outro lado, a criação de histórias de usuário ocorre manualmente, ficando a cargo do usuário desenvolvê-las, como no fluxo alternativo do Speech-To-Story. O JIRA Agile também não foi projetado para uso em entrevistas, e assim não oferece perguntas. Por fim, não se faz nenhuma análise ou processamento sobre as histórias geradas.

O JIRA, portanto, é utilizado para geração de histórias de usuário. Por outro lado, não realiza processamento de linguagem natural para gerar estas histórias, e apenas aceita quaisquer *inputs* manuais. Por fim, não é realizada nenhuma análise gramatical sobre a saída.

### *E. Agile Story Creator*

Para a criação de histórias de usuário em dispositivos móveis, a empresa AscentIQ desenvolveu o Agile Story Creator [40]. Esta ferramenta tem como objetivo a criação das histórias, seguindo o *template*.

Esta foi a única aplicação encontrada para tratar histórias de usuário de maneira nativamente *mobile*, e presente na *Google Play Store*. A utilização da ferramenta consiste no preenchimento, pelo engenheiro de requisitos, de cada campo correspondente no *template* da história de usuário.

Seu *input* de dados precisa ser manual, incluindo o título da funcionalidade elicitada. A história gerada pode ser exportada via *e-mail* para posterior manipulação. Como similaridade à solução proposta, o Agile Story Creator visa trazer uma abordagem móvel à geração de histórias de usuário, e suas funcionalidades também estão inclusas no Speech-To-Story, em fluxo alternativo.

Esta ferramenta correlata, apesar de tratar da geração de histórias, não realiza processamento de linguagem natural para criá-las, mas sim o exato dado que tenha sido definido pelo usuário. Finalmente, não é realizada nenhuma análise gramatical sobre a saída.

### *F. Comparativo*

A partir da Tabela III, comparativa, é possível observar pontos em comum entre as soluções, bem como as diferenças em relação ao tipo de *input* e documentação geradas. Ao mesmo tempo, é perceptível como cada uma das ferramentas correlatas não implementa as estruturas de processamento de linguagem natural com o mesmo intuito do Speech-To-Story de gerar histórias de usuário.

O RA foi pensado para uso em entrevistas de requisito e inclusive implementa *input* de respostas do usuário por voz. Porém, ele objetiva a geração de um documento de requisitos tradicionais, o que se configura como um limite deste para uso em ambientes ágeis.

O KBRA, por sua vez, realiza processamento de linguagem natural. Porém, seu uso não foi pensado para tempo de entrevista. Desta maneira, não é possível aplicá-lo para as mesmas circunstâncias.

Já o FAES possui uma estrutura mais similar ao Speech-To-Story, provendo novas perguntas para o engenheiro de requisitos conforme respostas vão sendo recebidas, através de processamento de linguagem natural. Porém, estas respostas são inseridas obrigatoriamente de forma manual, o que se configura como uma limitação. Outro impeditivo é que não existem documentações geradas por esta ferramenta. Por fim, ele não está inserido no contexto *mobile*.

JIRA e Agile Creator possuem *interfaces mobile*, e tratam da criação de documentos em formato de histórias de usuário, tal qual o Speech-To-Story. Porém, ambos são limitados pelo fato de que a criação de histórias é unicamente manual, o que limita o engenheiro a apenas digitar as respostas, na condução da entrevista, analisar seus resultados, e só então poder realizar a validação. A criação de histórias manualmente também está sujeita a falhas humanas, como erros de escrita e ambiguidades. Outra limitação é que tais ferramentas apenas preenchem *templates* de histórias de usuário, não realizando análise morfológica por linguagem natural, tampouco correções gramaticais sobre as histórias geradas.

O Speech-To-Story agrega as características de captação de respostas do cliente por voz presente no RA com processamento de linguagem natural sobre os resultados obtidos, tal

| | RA | KBRA | FAES | JIRA | Agile Story Creator | Speech-To-Story |
|---|---|---|---|---|---|---|
| Plataforma | Desktop | Desktop | Desktop | Web/**Mobile** | **Mobile** | **Mobile** |
| Provê perguntas | Não | Não | **Sim** | Não | Não | **Sim** |
| Usado em entrevistas | **Sim** | Não | **Sim** | Não | Não | **Sim** |
| Formato de Entrada | **Voz** | Manual | Manual | Manual | Manual | **Voz** |
| Documentação Gerada | Requisito Tradicional | Não Gera | Não Gera | **História** | **História** | **História** |
| Proc. Linguagem Natural | **Sim** | **Sim** | **Sim** | Não | Não | **Sim** |
| Correção Gramatical | Não | Não | Não | Não | Não | **Sim** |

Tabela III: Quadro comparativo entre as soluções.

qual o KBRA. Além disso, oferece novas perguntas ao engenheiro de requisitos conforme respostas vão sendo obtidas, como o FAES. Similar ao JIRA Agile e o Agile Story Creator, o Speech-To-Story visa trazer esta abordagem para o ambiente *mobile*, com o intuito de combinar estas características para prover geração automática de histórias. Esta solução vai além, adicionando processamento de linguagem natural para análise morfológica, com o intuito de gerar as histórias, e por fim, realiza correções gramaticais sobre seus produtos gerados.

## VI. Avaliação

### A. Contexto

Com o intuito de avaliar a solução proposta, fez-se necessário testá-la sobre um sistema real, coletando respostas contendo informações sobre este *software*, para ao final gerar histórias de usuário. Partindo desta premissa, optou-se pela condução de uma entrevista estruturada, contando apenas com o próprio Speech-To-Story para prover as perguntas, captar as respostas e criar as histórias.

Para esta validação, foi recebida a colaboração de uma *startup*, com foco em desenvolvimento de *software*, e escritório na capital baiana. Esta empresa atua no negócio de soluções jurídicas, captando informações de processos judiciais, de diários oficiais e tribunais ao redor do Brasil para disponibilizá-las de maneira simplificada e atualizada para o público em geral. A empresa permitiu que informações acerca de seu negócio fossem veiculadas neste trabalho, a partir da entrevista com um de seus colaboradores.

O entrevistado para esta validação é um dos colaboradores da empresa, e que já trabalha há anos no negócio de serviços jurídicos. Pelo fato de atuar na evolução dos sistemas da companhia, possui conhecimento acerca dos múltiplos fluxos que são executados, além de deter visibilidade em relação aos usos que os clientes fazem da plataforma.

O sistema objeto da entrevista é referenciado como Monitoramento Jurídico. Este é um assistente que permite aos seus usuários monitorar nomes e termos em diários oficiais, além de acompanhar o andamento de processos judiciais em tribunais. Para isso, os clientes possuem *interfaces* de personalização, de modo a parametrizar as informações que querem buscar, a frequência com que a ferramenta deve procurar por atualizações, além de filtrar quais as fontes nas quais deve-se procurar estes parâmetros. Para tanto, disponibiliza-se *interfaces* em *web* e *mobile*, além de planos de assinatura com funcionalidades adicionais.

### B. Experimento

A entrevista foi conduzida no dia 03 de maio de 2021, durando pouco menos de 2 horas. Para resguardar entrevistador e entrevistado, em face à pandemia de COVID-19, toda a conversa foi realizada de maneira virtual, pelo Google Meet.

A não-possibilidade do encontro presencial forçou o uso de diversos aparelhos em conjunto para viabilizar a realização da entrevista. A aplicação foi instalada em um aparelho Android Motorola X4, conectado à internet via Wi-Fi. A ligação via Google Meet, por sua vez, ocorreu em um segundo aparelho Android, Xiaomi Redmi 4X. Com o intuito de amplificar o áudio da conversa, de modo a tentar melhorar as capturas de voz, este segundo aparelho foi conectado via *bluetooth* a uma caixa de som JBL Clip 3. Desta forma, as perguntas ocorriam pelo microfone do aparelho celular Xiaomi, ao passo que as respostas eram ouvidas a partir da caixa de som JBL. O Motorola X4, contendo o Speech-To-Story, capturava a voz amplificada do entrevistado.

Embora não sendo as condições ideias, o uso desta configuração buscou manter as condições de entrevista pretendidas originalmente para o projeto: ter o entrevistador-entrevistado em comunicação direta, com a aplicação em posse do primeiro, e sem necessidade do segundo ter contato com a solução.

A entrevista foi iniciada com uma breve explicação, de modo a contextualizar o entrevistado acerca do trabalho que foi desenvolvido, a documentação que se desejava obter em formato de histórias e a imediata validação destas junto à pessoa indagada. A partir daí, para introduzí-lo à dinâmica de perguntas e captação de respostas da solução, foram realizadas questões introdutórias, opcionais para o fluxo da aplicação, de modo a melhor entender e documentar o perfil do entrevistado,

a sua empresa e o sistema em que trabalha. Por apenas objetivar uma aclimatação do entrevistado, essas respostas obtidas não são consideradas para a geração da documentação que se quer avaliar, e tampouco recebem qualquer tipo de tratamento ou avaliação morfológica. Todavia, elas são armazenadas em base e também constam nos arquivos exportados pelo Speech-To-Story.

Depois de concluída a fase introdutória, iniciou-se de fato o fluxo da entrevista. Para tanto, questionou-se sobre quem são as pessoas que manipulam o sistema de Monitoramento Jurídico. A resposta obtida foi condizente ao que foi dito pelo entrevistado e identificou-se que estes papéis são desempenhados por advogados, empresas e qualquer pessoa do âmbito geral que deseje monitorar processos judiciais. Por se tratar de um sistema que é abastecido de maneira automática, não há papéis realizados por funcionários da própria empresa para alimentar o serviço. Assim, os únicos atores identificados constam como se observa na Figura 23.

Uma vez identificados estes papéis, a entrevista continuou para elucidar as ações de cada um deles, a começar pelas pessoas em geral. Foi captado que a principal ação realizada por este público é a criação dos monitoramentos de um processo. A título de informação, estes monitoramentos são, internamente, *jobs* cujo objetivo é buscar atualizações nas diversas fontes dos processos e apresentar ao cliente caso elas existam. Para esta ação de criação, o cliente obtém múltiplas saídas: autos do processo, eventos que tenham ocorrido com aquele processo, além de uma linha do tempo agregando informações.

Destaca-se dentre as demais ações possíveis para este usuário, que foram captadas pela solução, a possibilidade de geração de relatórios. Similar à ação anterior, também resulta-se em múltiplas saídas, sendo elas a visualização de monitoramentos por estado e acompanhamento da evolução dos processos. Também foram obtidas ações de gestão desses marcadores e parametrização deles de acordo com a necessidade do usuário.

O segundo papel questionado, de advogados, teve como ações a busca de termos dentro dos resultados de monitoramento e envio de *feedbacks* para a empresa através da ferramenta. Suas saídas também foram obtidas, na sequência. Por fim, foram realizadas as perguntas relacionadas ao papel das empresas. Estas realizariam ações de assinar serviços da companhia, em diferentes planos, e obter os benefícios de cada um daqueles pacotes. Finalizadas as perguntas, foram geradas as histórias de usuário, imediatamente compartilhadas em formato PDF com o entrevistado para validação, conforme Figura 24.


Figura 23: Papéis identificados na avaliação.


Figura 24: Ações realizadas pelo papel de "advogados".

### C. Discussão e resultados obtidos

A entrevista ocorreu como uma conversa natural, para que o entrevistado não se sentisse forçado a responder conforme apenas a especificidade da ferramenta. A partir das respostas coletadas na conversa, foram geradas 22 histórias de usuário, após 53 segundos de processamento. Este curto tempo para geração de histórias indica que a solução foi capaz de atender à ideia de produzir documentação imediatamente após a entrevista, de maneira automatizada. Cada papel recebeu histórias contendo as ações que realizam sobre o sistema, e os resultados que deseja obter ao realizá-las. Para as ações que possuem mais de uma saída, histórias específicas foram geradas para cada um desses resultados, assim como é evidenciado na Figura 25.


Figura 25: Exemplos de histórias de usuário geradas a partir da entrevista de validação.

Ao compartilhar essa documentação com o entrevistado, foi possível que este fizesse a validação imediata das histórias, também cumprindo com outro dos objetivos da aplicação. Nesta validação, não foram encontrados erros na avaliação morfológica, indicando que as histórias geradas contém as classes gramaticais necessárias para preencher os templates de história de usuário. Por outro lado, esta validação identificou erros de concordância verbal, decorrentes de limitações da própria solução. A validação seguiu normalmente, uma vez que estes percalços não se mostraram impactantes ao nível de impedir a continuação da análise, sugerindo que as histórias faziam sentido do ponto de vista da compreensão humana. Além disso, todas as histórias condiziam com as respostas que foram obtidas, refletindo assim os requisitos do sistema de Monitoramento Jurídico para cada papel.

Ainda atingindo outra premissa da solução, ao final desta validação, puderam ser realizadas correções logo que os erros foram identificados. A ferramenta armazenou os ajustes em banco, e após nova exportação, as histórias corrigidas estavam em posse do cliente para uma segunda rodada de validação. Todo este processo de correção, persistência e nova exportação durou cerca de 6 minutos. Nesta nova análise, nenhum problema adicional foi observado nas histórias, e o fluxo do teste foi concluído.

É preciso, porém, ressaltar limitações desta entrevista. A mais notória é a efetividade da captação de respostas por voz: em testes unitários com o aparelho em mãos, o microfone do celular identifica as palavras ditadas com maior qualidade. Porém, com limitações decorrentes da necessidade de isolamento social, algumas capturas de voz resultaram em palavras identificadas erroneamente. Para cobrir essas eventuais falhas e limitações, o Speech-To-Story fornece uma *interface* para editar estes possíveis erros. Uma avaliação contendo um panorama mais preciso da efetividade da captura por voz só será possível em um cenário pós-pandêmico.

Outra limitação é na concordância verbal das histórias de usuário criadas pela solução. A implementação de uma análise para checar e adequar possíveis erros deste tipo não estava no escopo original no projeto, e iria requerer mais tempo de estudo e desenvolvimento. Além desta melhoria futura, outra possibilidade correlata é a adequação dos tempos verbais nas histórias para primeira pessoa, o que hoje é outra limitação do trabalho. Estes fatores, se solucionados, fariam com que o entrevistado pudesse receber histórias mais similares ao modelo que se utiliza na indústria.

Por fim, outro fator limitante na entrevista foi a necessidade de estar *online* e conexão com à internet. Por algumas vezes, a conexão do entrevistado foi interrompida e a conferência foi paralisada. Isso não afetou os resultados da aplicação, pois a solução suporta interrupções e continuações. Porém, as frequentes pausas desaceleraram o ritmo da entrevista, forçando repetições e re-contextualizações.

Como resultado da entrevista realizada para validação, algumas das histórias de usuário geradas sobre a solução de Monitoramento Jurídico estão disponíveis no apêndice.

## VII. Conclusão

A partir dos estudos expostos, pode-se compreender que a utilização de documentação robusta em ambientes ágeis não é uma prática encorajada, já que uma das premissas destas metodologias é o uso de mínimos documentos. Para estas abordagens, produz-se melhores resultados com histórias de usuário, que quebram os requisitos em casos de uso, de maneira a nortear a equipe de desenvolvimento.

Observa-se, porém, problemas no processo de criação destas histórias, uma vez que todo o processo de entrevista, análise e escrita é feito de forma manual, estando sujeito a falhas humanas ao longo destes passos. O efeito disso é que a validação das histórias geradas só pode ser realizada em momentos posteriores à entrevista, o que pode gerar ainda mais re-trabalho.

Conclui-se, com a solução exposta, que é possível atacar estes problemas através de uma abordagem *mobile*, utilizando processamento de linguagem natural, para automatizar o processo de criação de histórias. O Speech-To-Story foi capaz de fornecer perguntas para uso ao longo da entrevista, capturar as respostas do entrevistado, gerar histórias de usuário condizentes com o que foi obtido, e por fim validá-las ainda em tempo de entrevista junto ao entrevistado.

É possível ainda concluir que o aproveitamento de ferramentas CASE em plataformas móveis pode se caracterizar como uma área a ser explorada, em função de sua versatilidade, recursos de *hardware* e facilidade de manipulação. A solução desenvolvida já é capaz de entregar histórias com sucesso a partir da entrada do usuário. Para além disso, porém, o Speech-To-Story aponta para um campo ainda pouco explorado e que pode configurar em evoluções ainda maiores para resolver outros desafios relacionados à geração de documentação de maneira automática.

### A. *Limitações deste Trabalho*

Existem fatores limitantes em torno da solução desenvolvida, que fazem com que esta ainda possa receber melhorias futuras. Tais limitações se configuram principalmente como estruturais, em torno de dependências e dispositivos.

Inicialmente, a solução se baseia em uma biblioteca de processamento de linguagem natural de terceiros. Embora o OpenNLP seja *open-source* e atualizado pela Apache, ele ainda depende da manutenção da empresa, estando fora do escopo de desenvolvimento do Speech-To-Story.

Outra limitação é a necessidade de estar sempre *online* no momento da realização das entrevistas. Como as respostas obtidas são enviadas ao servidor da Google para transcrição, o aparelho em que o Speech-To-Story executa precisa de constante conexão com a internet. Este é o único empecilho que impede a execução completa de maneira offline.

Uma terceira limitação é a necessidade de se utilizar um dispositivo Android para execução da solução. O código foi desenvolvido em Xamarin, o que confere um caráter multiplataforma. Porém, diante da indisponibilidade de um dispositivo iOS, esta aplicação foi apenas testada e evoluída em aparelhos Android.

Nos testes realizados, encontrou-se dificuldade em utilizar a ferramenta em entrevistas remotas. Isto porque a captura das respostas, neste caso, depende da qualidade do dispositivo de som que emite o áudio, bem como do microfone do aparelho que captará este áudio. Desta forma, está pendente a validação de maneira presencial. É possível que, eliminados os ruídos citados, os resultados possam ser até mesmo melhores.

### B. *Trabalhos Futuros*

Com o Speech-To-Story atuando como ferramenta CASE, abrem-se possibilidades de trabalhos futuros, como melhorias à solução. Desta forma, seria possível incrementar o que foi desenvolvido, dando capabilidades que estendem as funções atuais.

A ferramenta, atualmente, não contempla a identificação de histórias de usuário longas ou genéricas. É desejável que as histórias sejam curtas e detalhadas, de modo que um possível incremento seria a adição desta inteligência para alertar o usuário e, se possível, oferecer para que se destrinche aquela história em outras menores.

Também é desejável que futuramente a ferramenta possa atuar diretamente em situações de estrangeirismos, contrações e gírias, com a finalidade de apresentar sinônimos ao engenheiro de requisitos ainda durante a captura das respostas. Esta limitação, hoje, faz com que exista a possibilidade das histórias serem geradas apresentando uma linguagem informal, forçando que o usuário seja o responsável por editar e adequar à norma formal.

Outra melhoria a se fazer está relacionada à abordagem de entrevistas estruturadas: é desejável que exista flexibilidade em relação às perguntas que são oferecidas ao usuário. Na solução atual, as perguntas são fixas e não se alteram de acordo com o contexto do entrevistado. Porém, entrevistas estruturadas que contém perguntas adaptáveis ao negócio podem induzir a mais respostas, gerando mais histórias.

Por fim, um possível incremento é uma mudança na abordagem de utilização do Speech-To-Story. Ao invés de realizar a captura de cada pergunta, seria possível que a ferramenta captasse a entrevista inteira para que, no final, aquela captura bruta fosse analisada e gerasse as histórias. Tal solução necessitaria da implementação de uma inteligência que atuasse sobre os resultados, de maneira a identificar neles quem são as perguntas, quais são suas resposta e então classificá-las.

## REFERÊNCIAS


[1] R. S. Corporation, ""rational unified process": Best practices for software development teams, tpb026b, rev. 11/01," *Rational Unified Process (RUP) (2001)*, 11 2001. [Online]. Available: http://www.therationaledge.com.

[2] E. W. Dijkstra, "The humble programmer," *Commun. ACM*, vol. 15, no. 10, p. 859–866, Oct. 1972. [Online]. Available: https://doi.org/10.1145/355604.361591

[3] "Ieee standard glossary of software engineering terminology," *IEEE Std 610.12-1990*, pp. 1–84, 1990.

[4] P. Naur and B. Randell, *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO*, 1969.

[5] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, "Manifesto for agile software development," 2001.

[6] S. Hastie Wick and Angela, "User stories and use cases - don't use both!" https://www.batimes.com/articles/user-stories-and-use-cases-dont-use-both.html, 2004 (acessado 1 de Agosto, 2018).

[7] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2015.

[8] P. A. Laplante and C. J. Neill, "The demise of the waterfall model is imminent," *Queue*, vol. 1, no. 10, p. 10, 2004.

[9] J. Highsmith, "Agile software development ecosystem," 01 2002.

[10] M. Cohn, *User Stories Applied: For Agile Software Development*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.

[11] A. M. Hickey, A. M. Davis, and D. Kaiser, "Requirements elicitation techniques: Analyzing the gap between technology availability and technology use," *Comparative Technology Transfer and Society*, vol. 1, no. 3, pp. 279–302, 2003.

[12] A. Ferrari, P. Spoletini, and S. Gnesi, "Ambiguity cues in requirements elicitation interviews," pp. 56–65, 09 2016.

[13] T. E. Bell and T. A. Thayer, "Software requirements: Are they really a problem?" in *Proceedings of the 2nd international conference on Software engineering*. IEEE Computer Society Press, 1976, pp. 61–68.

[14] D. E. H. Damian, "Challenges in requirements engineering," 2000.

[15] J. Brugger, "Importance of interview and survey questions in systems analysis," Citeseer, Tech. Rep., 2010.

[16] E. A. Isaacs, "Interviewing customers: discovering what they can't tell you," in *CHI'97 Extended Abstracts on Human Factors in Computing Systems*. ACM, 1997, pp. 180–181.

[17] D. Carrizo, O. Dieste, and M. López, "Identifying moderator variables through requirements elicitation experiments limitations," in *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement*, ser. Profes '11. New York, NY, USA: ACM, 2011, pp. 22–25. [Online]. Available: http://doi.acm.org/10.1145/2181101.2181107

[18] O. Dieste and N. Juristo, "Systematic review and aggregation of empirical studies on elicitation techniques," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 283–304, 2011.

[19] B. L. Berg, H. Lune, and H. Lune, *Qualitative research methods for the social sciences*. Pearson Boston, MA, 2004, vol. 5.

[20] A. P. P. Gilvaz and J. S. do Prado Leite, "Faes: A case tool for information acquisition," in *Computer-Aided Software Engineering, 1995. Proceedings., Seventh International Workshop on*. IEEE, 1995, pp. 260–269.

[21] J. C. S. d. P. Leite and A. P. P. Gilvaz, "Requirements elicitation driven by interviews: The use of viewpoints," in *Proceedings of the 8th International Workshop on Software Specification and Design*, ser. IWSSD '96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 85–. [Online]. Available: http://dl.acm.org/citation.cfm?id=857204.858259

[22] M. Fowler, "What is the difference between a usecase and xp's userstory," https://www.martinfowler.com/bliki/UseCasesAndStories.html, 2003 (acessado 1 de Agosto, 2018).

[23] B. Wake, "Invest in good stories," https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/, 2003 (acessado 1 de Agosto, 2018).

[24] S. Tiwari and S. S. Rathore, "A methodology for the selection of requirement elicitation techniques," *arXiv preprint arXiv:1709.08481*, 2017.

[25] R. Mall, *Fundamentals of Software Engineering*, 4th ed. Prentice-Hall of India Pvt.Ltd, 2014.

[26] W. S. Humphrey, "Case planning and the software process," in *Case Technology*. Springer, 1991, pp. 59–75.

[27] "Mobile operating system market share worldwide," https://gs.statcounter.com/os-market-share/mobile/worldwide, accessed: 2020-07-04.

[28] "Number of apps available in leading app stores as of 1st quarter 2020," https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, accessed: 2020-07-04.

[29] "What is xamarin?" https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin, accessed: 2020-07-04.

[30] "What is sqlite?" https://www.sqlite.org/index.html, accessed: 2020-07-04.

[31] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. USA: Prentice Hall Press, 2009.

[32] M. A. Covington, *Natural Language Processing for PROLOG Programmers*, 1st ed. USA: Prentice Hall PTR, 1993.

[33] "Chapter 6. part-of-speech tagger," https://opennlp.apache.org/docs/1.8.0/manual/opennlp.html#tools.postagger, accessed: 2020-07-04.

[34] "Welcome to apache opennlp," https://opennlp.apache.org, accessed: 2020-07-04.

[35] M. A. Covington, *Natural Language Processing for PROLOG Programmers*, 1st ed. USA: Prentice Hall PTR, 2008.

[36] W. de Pádua Paula Filho, *Engenharia de software*, 2nd ed., 2003.

[37] H. B. Reubenstein and R. C. Waters, "The requirements apprentice: automated assistance for requirements acquisition," *IEEE Transactions on Software Engineering*, vol. 17, no. 3, pp. 226–240, March 1991.

[38] D. R. Harris and J. A. J. Czuchry, "Kbra: A new paradigm for requirements engineering," *IEEE Intelligent Systems*, vol. 3, pp. 21–24, 26–32, 34–35, 12 1988. [Online]. Available: doi.ieeecomputersociety.org/10.1109/64.10017

[39] "Jira agile," https://www.atlassian.com/software/jira/agile#:~:text=Jira%20Software%20is%20an%20agile,projects%20from%20a%20single%20tool., accessed: 2020-07-04.

[40] "Agile story creator," https://play.google.com/store/apps/details?id=com.ascentiqsolutions.agilestory&hl=en&gl=US, accessed: 2020-07-04.

APÊNDICE A
RESULTADOS

Validação TCC
ID: [1] Por: Entrevistado

Como pessoas em geral, eu gostaria de criar monitoramentos de um processo, pessoa, empresa ou advogado, para preencher um formulário com dados da empresa ou pessoa que quer monitorar

Validação TCC
ID: [2] Por: Entrevistado

Como pessoas em geral, eu gostaria de criar monitoramentos de um processo, pessoa, empresa ou advogado, para ter acesso aos autos do processo

Validação TCC
ID: [3] Por: Entrevistado

Como pessoas em geral, eu gostaria de criar monitoramentos de um processo, pessoa, empresa ou advogado, para obter os eventos daquele processo

Validação TCC
ID: [4] Por: Entrevistado

Como pessoas em geral, eu gostaria de criar monitoramentos de um processo, pessoa, empresa ou advogado, para obter uma timeline contendo informações agregadas

Validação TCC
ID: [5] Por: Entrevistado

Como pessoas em geral, eu gostaria de obter relatórios, para ver monitoramentos por estado

Validação TCC
ID: [6] Por: Entrevistado

Como pessoas em geral, eu gostaria de obter relatórios, para acompanhar a evolução dos processos

Validação TCC
ID: [7] Por: Entrevistado

Como pessoas em geral, eu gostaria de obter relatórios, para obter os tipos de monitoramento

Validação TCC

ID: [8] Por: Entrevistado

Como pessoas em geral, eu gostaria de organizar os monitoramentos em pastas, para obter uma lista de monitoramentos de nome, processos e pessoas

Validação TCC

ID: [12] Por: Entrevistado

Como pessoas em geral, eu gostaria de adicionar variações no monitoramento, para ampliar as possibilidades de busca

Validação TCC

ID: [9] Por: Entrevistado

Como pessoas em geral, eu gostaria de clicar nos marcadores, para listar apenas os monitoramenros que tem a ver com aquele marcador

Validação TCC

ID: [13] Por: Entrevistado

Como pessoas em geral, eu gostaria de adicionar variações no monitoramento, para aumentar a quantidade de resultados obtidos

Validação TCC

ID: [10] Por: Entrevistado

Como pessoas em geral, eu gostaria de adicionar e-mails ao monitoramento, para ser notificado de mudanças no monitoramento

Validação TCC

ID: [14] Por: Entrevistado

Como pessoas em geral, eu gostaria de adicionar regras, para definir termos que devem conter nos resultados

Validação TCC

ID: [11] Por: Entrevistado

Como pessoas em geral, eu gostaria de escolher as fontes de monitoramento, para listar dados apenas da fonte desejada

Validação TCC

ID: [15] Por: Entrevistado

Como pessoas em geral, eu gostaria de adicionar regras, para termos que não devem conter nos resultados

Validação TCC

ID: [16] Por: Entrevistado

Como um advogado, eu gostaria de buscar termos dentro dos resultados do monitoramento, para obter uma lista com os monitoramentos contendo aquele termo

Validação TCC

ID: [17] Por: Entrevistado

Como um advogado, eu gostaria de enviar feedback para o escavador, para preencher um formulário com informações

Validação TCC

ID: [18] Por: Entrevistado

Como um empresa, eu gostaria de assinar o serviço do escavador, para obter serviços diferenciados

Validação TCC

ID: [19] Por: Entrevistado

Como um empresa, eu gostaria de assinar o plano básico, para ganhar 3 monitoramentos do processo

Validação TCC

ID: [20] Por: Entrevistado

Como um empresa, eu gostaria de assinar o plano básico, para resultados ilimitados referentes aos monitoramentos

Validação TCC

ID: [21] Por: Entrevistado

Como um empresa, eu gostaria de assinar o padrão, para acesso aos documentos públicos do processo e frequência de atualização diária

Validação TCC

ID: [22] Por: Entrevistado

Como um empresa, eu gostaria de assinar o plano avançado, para acesso a 50 monitoramentos e atualização diária dos processos