UM ESTUDO SOBRE O USO DO FRAMEWORK COMBINE ATRAVÉS DA MINERAÇÃO DE PUBLICAÇÕES DO STACK OVERFLOW

A study about the usage of the Combine framework using mining on Stack Overflow

Alessandra Luana Nascimento Pereira
Centro de Informática – Universidade Federal de Pernambuco (UFPE)

RESUMO

Nos dias de hoje são utilizadas diversas aplicações no cotidiano das pessoas, tais como aplicações *web*, aplicações móveis, ou mesmo aplicações em videogames ou em dispositivos embarcados. Muitas delas são sistemas desenvolvidos com base no uso da programação reativa. Quando comparadas as aplicações desenvolvidas com paradigmas imperativos, esses sistemas reativos conseguem ser mais flexíveis, menos acoplados e mais escaláveis, resultando na redução de diversos custos. Dentre eles estão os de implementação, manutenção e custos no tempo de resposta. Por esses e outros motivos, existe uma alta popularidade do uso de arquiteturas baseadas em *frameworks* que aplicam a programação reativa no desenvolvimento de aplicações móveis. Sendo *Combine* um *framework* lançado em 2019 pela empresa multinacional norte-americana de tecnologia Apple, esse *framework* apresenta um suporte nativo de maneira explícita ao uso do paradigma reativo no desenvolvimento de aplicações em *Swift* para dispositivos do ecossistema da empresa. Oferecendo melhor performance e dando assim o poder de descartar o uso de outras soluções e dependências de terceiros que antes faziam isso. Dessa maneira, o presente estudo realiza uma análise sobre as maiores dificuldades relacionadas ao uso do *framework Combine*, baseando-se na mineração de dados pelo *Stack Overflow* utilizando modelagem de tópicos com a execução de LDA, a fim de identificar e entender as maiores dificuldades que os desenvolvedores *Swift* estão mais propensos a enfrentar ao desenvolver aplicações com o *framework*. Como resultado foi possível extrair 20 tópicos associados às dúvidas publicadas no *Stack Overflow* relacionadas ao seu uso e identificar os tópicos mais frequentes, mais populares e mais difíceis segundo as métricas definidas no estudo. Acredita-se que os resultados aqui apresentados podem não apenas servir de fonte para outras pesquisas relacionadas ao mesmo, mas também entender os principais problemas enfrentados pelos desenvolvedores e ajudar aqueles que estão aprendendo a utilizá-lo ou mesmo já utilizam em suas aplicações.

Palavras-chave: programação reativa; combine; modelagem de tópicos; LDA.

ABSTRACT

Nowadays, several applications are used in people's daily lives, such as web applications, mobile applications, or even applications in video games or embedded devices. Many of them are systems developed based on the use of reactive programming. When compared to applications developed with imperative paradigms, these reactive systems can be more flexible, less coupled, and more scalable, thus resulting in the reduction of several costs, among them are the implementation, maintenance, and costs in response time. For these and other reasons, there is high popularity of using architectures based on frameworks that apply reactive programming in the development of mobile applications. As Combine is a framework launched in 2019 by the American multinational technology company Apple, this framework explicitly supports the use of the reactive paradigm in Swift application development for

devices in the company's ecosystem. Offering better performance and thus giving the power to discard the use of other solutions and third-party dependencies that used to do this. In this way, the present study performs an analysis of the major difficulties related to the use of the Combine framework, based on data mining by Stack Overflow using topic modeling with the execution of LDA, to identify and understand the key difficulties that Swift developers are more likely to face when developing applications with the framework. As a result, it was possible to extract 20 topics associated with the questions published on Stack Overflow related to its use and identify the most frequent, most popular, and most difficult topics according to the metrics defined in the study. It is believed that the results presented here can not only serve as a source for other research related to it but also understand the major problems faced by developers and help those who are learning to use it or even already use it in their applications.

Keywords: reactive programming; combine; topic modeling; LDA.

1 INTRODUÇÃO

Atualmente são utilizadas diversas aplicações no nosso cotidiano, aplicações essas que vão desde as web, como também as presentes em dispositivos móveis, ou até mesmo em videogames e dispositivos embarcados. Muitas dessas aplicações são sistemas reativos, ou seja, eles respondem a ocorrências de eventos e com isso realizam ações, que podem também desencadear novos eventos. Como exemplo mais concreto é possível citar as aplicações que possuem uma interface gráfica e respondem ao toque do seu usuário [1]. Outra característica desses sistemas que é relevante é a capacidade de processar dados que são originados em locais ou de fontes diversas [2], como banco de dados, memória ou o próprio usuário. Da mesma forma que esse processamento pode ocorrer de maneira simultânea e são emitidas ao longo do tempo de uma maneira não previsível. O que torna sua aplicação pertinente quando consideramos o contínuo avanço da tecnologia e as distintas capacidades dos sistemas atuais.

É nesse contexto que a programação reativa (PR) entra. É possível defini-la como um paradigma de programação declarativo que está relacionado aos fluxos de dados, logo, os valores contínuos que variam no tempo e à propagação de mudanças de forma assíncrona [2]. Dessa maneira, os sistemas que são desenvolvidos com base nesse paradigma conseguem ser mais flexíveis, menos acoplados e mais escaláveis, reduzindo, assim, os custos de implementação, de tempo de resposta e de manutenção quando comparados a paradigmas imperativos [3].

Devido ao interesse existente de modo crescente por parte dos seus praticantes e pela comunidade de linguagens de programação [1], muitas soluções foram desenvolvidas para ajudar demais linguagens a funcionarem de maneira mais reativa. Esses recursos visam fornecer aos desenvolvedores um maior controle sobre o fluxo de dados, já que em programas reativos ocorre uma inversão de controle, no qual o fluxo de controle é guiado por eventos externos e não pela ordem especificada em seu código fonte pelo programador [2].

Podemos citar como exemplo de solução que auxilia o paradigma a *Reactive Extensions*[1], também conhecida como *ReactiveX* ou *Rx*. Sendo ela uma coleção de projetos *open source* disponibilizado como uma das bibliotecas reativas mais populares e amplamente

[1] Reactive Extensions. Disponível em: https://reactivex.io/. Acesso em: 07 de outubro de 2022.

utilizada por diversas linguagens de programação[2]. Além das populares bibliotecas de *front-end* como *React.js*, *Bacon.js* e *Knockout* [1].

Existe também uma alta popularidade no uso de arquiteturas baseadas em *frameworks*[3] que aplicam a programação reativa no desenvolvimento de aplicativos móveis [4]. Tendo em vista que os desenvolvedores desse tipo de plataforma precisam lidar com a crescente necessidade de tratar várias tarefas simultâneas, como, por exemplo, durante a reprodução de um áudio, realizar requisições de rede ou gerir toques e gestos na interface. Passar essas informações e dados de um processo para outro, mesmo quando acontecem de maneira simples e numa sequência correta assincronamente, podem acabar gerando vários problemas [5].

Para o contexto do desenvolvimento de aplicativos móveis na linguagem de programação *Swift*[4], podemos citar algumas dessas soluções: *RxSwift*[5], *PromiseKit*[6] e o mais recente deles, o *Combine* [4]. Segundo informações de desenvolvedores que utilizam tais soluções, e com base na análise da documentação fornecida pela Apple, o *framework Combine* pode ser considerado um estágio lógico na evolução do paradigma de programação reativa no contexto do desenvolvimento de aplicações em *Swift* [4].

Sua principal vantagem sobre os demais concorrentes é que ele é desenvolvido pela própria empresa americana Apple, ou seja, são os próprios engenheiros da Apple que a desenvolvem, logo eles possuem acesso a APIs internas que não estão disponíveis para terceiros fora dela. Da mesma maneira que possuem conhecimento profundo e privilegiado sobre os dispositivos de hardware aos quais as aplicações que usam *Combine* serão executadas. Isto permite que o *framework* seja otimizado e que apresente vantagens em seu uso em relação aos demais, por exemplo: possuir tempos de resposta mais rápidos, ou mesmo exercer menor pressão sobre o uso da bateria e ter um consumo mais otimizado [4].

Sendo um *framework* relativamente novo, já que foi lançado em 2019, ainda existe pouca evidência de pesquisas relacionadas ao seu uso. Da mesma maneira que existe uma necessidade de mais estudos alinhados com a capacidade de aprendizado e compreensão por novos desenvolvedores ou mesmo por aqueles que já possuem experiência na construção de aplicações móveis que utilizam *frameworks* de programação reativa.

Assim, o presente trabalho busca investigar quais as maiores dificuldades que os utilizadores do *Combine* estão mais propensos a enfrentar ao desenvolver aplicações com o *framework*, com o objetivo de entender quais são esses obstáculos para possibilitar formas de mitigação por parte dos desenvolvedores e como também servir de fonte para outras pesquisas relacionadas ao mesmo.

Para isso, foi realizado um estudo no *Stack Overflow*[7] (SO) na expectativa de responder à seguinte Questão de Pesquisa (QP): Quais são as principais dúvidas que os desenvolvedores possuem ao utilizarem o *framework Combine* na construção de aplicações?

---

[2] Linguagens ReactiveX. Disponível em: http://reactivex.io/languages.html. Acesso em: 07 de outubro de 2022.

[3] Conjunto de códigos abstratos com objetivo de prover novas funcionalidades e recursos.

[4] Linguagem *Swift*. Disponível em: https://www.swift.org/. Acesso em: 07 de outubro de 2022.

[5] *RxSwift*. Disponível em: https://github.com/ReactiveX/RxSwift. Acesso em: 07 de outubro de 2022.

[6] *PromiseKit*. Disponível em: https://github.com/mxcl/PromiseKit. Acesso em: 07 de outubro de 2022.

[7] *Stack Overflow*. Disponível em: https://stackoverflow.com. Acesso em: 06 de outubro de 2022.

Esse estudo se deu por meio da mineração de dados do SO, por ser a maior plataforma colaborativa de perguntas e respostas e mais amplamente utilizada para o desenvolvimento de *software* e programação.

O restante do trabalho está organizado da seguinte maneira: na Seção 2 é apresentado o referencial teórico com o objetivo de contextualizar as áreas e temas abordados no estudo; a metodologia aplicada durante o processo de coleta e mineração dos dados do SO, detalhamento das suas etapas e as decisões tomadas são apresentadas na Seção 3; a Seção 4 tem como objetivo apresentar os resultados obtidos pela aplicação da metodologia e suas possíveis implicações; e por fim, a Seção 5 aborda as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção tem como objetivo detalhar os tópicos e temas abordados no presente estudo. Iniciamos tratando sobre a programação reativa, seguindo para uma melhor contextualização do *framework Combine*, e por fim, falamos de maneira breve sobre mineração do *Stack Overflow* e da modelagem de tópicos com o uso do *Latent Dirichlet Allocation* (LDA) para a extração de informações [6].

2.1 Programação Reativa

Sistemas reativos simbolizam uma extensa e abundante parcela de sistemas de *software* que precisam responder a eventos, sejam eles internos ou externos, com uma ação e que possivelmente geram novos eventos em si [7]. Em outras palavras, escrever um trecho de código reativo significa programar respostas a ações que irão acontecer em um determinado momento e que podem depender de valores e ações anteriores.

Como exemplo, considere uma aplicação que filtra uma tabela com base no conteúdo de um campo de texto. Cada tecla pressionada no campo de texto pode ser considerada como um evento, ao qual é possível realizar manipulações ou mesmo gerar novos eventos e usar o resultado para atualizar a interface do usuário da aplicação [8].

Não apenas como exposto no exemplo anterior, mas a PR está presente em diversas outras aplicações, como nas notificações que são recebidas em dispositivos móveis, ou mesmo em sistemas embarcados que respondem aos sinais de sensores que reagem a mudanças no ambiente [1]. Esses sistemas são orientados a mudanças que podem ocorrer ao longo do tempo nos seus fluxos de dados, e precisam responder a elas de uma maneira assíncrona. Dessa maneira, uma forma de possibilitar o desenvolvimento desses sistemas é utilizando a programação reativa, que é definida como um paradigma de programação declarativo que está relacionado aos fluxos de dados, logo, os valores contínuos que variam no tempo e à propagação de mudanças de forma assíncrona [2].

Mesmo havendo diferenças entre as linguagens de programação que utilizam o paradigma, os princípios seguem os mesmos para uma grande maioria. São eles: a programação declarativa, onde os desenvolvedores descrevem com relação às suas funcionalidades como os componentes dependem uns dos outros; abstração na propagação de mudanças, ou seja, não existe uma necessidade de atualizar manualmente valores dependentes, já que essa propagação de mudança se torna algo implícito em aplicações reativas; uso da composição, já que devido às abstrações é possível compor fluxos de dados mais complexos se necessário; e o favorecimento do fluxo de dados sobre o fluxo de controle, como já dito anteriormente, ocorre uma inversão de controle, onde o sistema reage a novos

dados e eventos em vez da execução seguindo o respectivo trecho de código produzido na aplicação [7].

Esses princípios estão presentes em diversas soluções que possibilitam o desenvolvimento reativo em linguagens também imperativas. Um desses exemplos, como já citado, é o *Reactive Extensions*. Uma das bibliotecas mais populares e utilizadas, conhecida também como *ReactiveX* ou *Rx*. Através dela, várias linguagens de programação podem funcionar de uma maneira mais reativa, como é o caso de uso para a linguagem *Swift* através do *RxSwift*.

Devido à conhecida complexidade de se construir as aplicações reativas, por conta dessa combinação mista entre os dados e o fluxo de controle, está a relevância do uso da programação reativa, uma vez que construir esses sistemas acaba por vezes se tornando mais simples quando o paradigma é utilizado [1]. Tanto que nos últimos anos houve um expressivo crescimento de interesse por pesquisadores e profissionais da área, devido a possibilidade de expressar comportamentos reativos complexos de uma maneira mais simples, intuitiva e declarativa [7].

2.2 Combine

O *Combine* é um *framework* de processamento de eventos assíncronos criado pela Apple em 2019 com o intuito de fornecer uma API declarativa para processar valores ao longo do tempo possibilitando, assim, mais uma forma de desenvolvimento reativo em *Swift* [9]. A relevância do seu uso comparado com os seus concorrentes é evidente, uma vez que ele é desenvolvido diretamente pela Apple, ou seja, os programadores responsáveis pelo seu desenvolvimento possuem acesso a APIs internas, as quais não estão disponíveis para outros desenvolvedores ou soluções de terceiros, como também possuem vasto conhecimento sobre os dispositivos de hardware que executam essas aplicações com o uso do *framework*. Devido a isso, é fornecido um alto nível de otimização nos níveis de processador e RAM, o que torna o tempo de resposta mais rápido ao utilizar o *Combine* significativamente e otimiza o consumo de bateria do dispositivo em contrapartida a outras soluções [4]. Com sua construção, agora é possível programar de maneira reativa para *Swift* sem a necessidade de bibliotecas externas como o *RxSwift*, *ReactiveSwift*[8] e *PromiseKit*, já que com o uso do *Combine* é possível ter a mesma experiência de uma forma nativa.

Seu funcionamento baseia-se no padrão *publishers/subscribers* [9]. Assim, temos um *publisher* que produz valores e os *subscribers* que irão consumir o valor produzido anteriormente [26]. Existe também um terceiro participante neste processo, os operadores. Estes são apenas um tipo diferente de *publisher*, sendo responsáveis por recolher as entradas que recebem valor de outro publicador para, então, produzir valores diferentes utilizando algumas operações do modelo pub/sub.

O *publisher* é um protocolo que declara um tipo de dado que pode transmitir uma sequência de valores ao decorrer do tempo [9]. Quando solicitado, o *publisher* pode fornecer dados se estes estiverem disponíveis. É importante ter em mente que um *publisher* é descrito com dois tipos associados a ele: o *output*, que corresponde ao tipo de valor que ele pode produzir, e o *failure*, o tipo de erro que pode ser encontrado [27].

Os operadores são métodos chamados por *publishers* e retornam para eles. São utilizados para manipular os valores, seja alterando, adicionando, transformando, combinando

---

[8] *ReactiveSwift*. Disponível em: https://github.com/ReactiveCocoa/ReactiveSwift. Acesso em: 07 de outubro de 2022.

ou mesmo filtrando-os. Além de possibilitar o encadeamento de múltiplos operadores, ou seja, uma operação pode requerer um outro operador para o funcionamento adequado, e é possível encadeá-los. Como exemplo nas chamadas de serviços de *web*: uma vez que a resposta do servidor é recebida, são realizadas algumas operações nessa resposta específica, mapeando e transformando-a.

Já o *subscriber* não é muito diferente. É um protocolo que declara um tipo o qual pode receber *input* de um *publisher*. Ele também possui dois tipos associados: o *input*, que é o tipo de valor que ele vai receber, e o *failure*, que necessita ser equivalente ao tipo de *failure* do *publisher*. Para atingir os objetivos esperados, o *publisher* e o *subscriber* precisam funcionar de forma sincronizada e para isso acontecer é necessário que o *output* tenha o mesmo valor do *input*, assim como o *failure*.

Este padrão *publisher-subscriber* é conhecido e utilizado além do desenvolvimento móvel, e sendo agora também possível de ser utilizado por desenvolvedores de aplicações em *Swift* de uma forma nativa com o uso do *Combine*.

2.3 Mineração no Stack Overflow

O *Stack Overflow*[9] é uma comunidade *online* de perguntas e respostas amplamente conhecida por profissionais e não-profissionais da área de desenvolvimento de *software*. Nele os desenvolvedores procuram sanar dúvidas, aprendendo e compartilhando conhecimento sobre os mais diversos tópicos envolvidos durante o desenvolvimento ou a manutenção de sistemas de software. Similar a maioria dos sites voltados a perguntas e respostas, no SO os usuários publicam questionamentos a respeito do universo do desenvolvimento de *software*, e respondem perguntas, comentam em publicações e votam.

Por ser amplamente utilizado por desenvolvedores das mais diversas áreas ou temas, os conteúdos produzidos pela comunidade *online* possuem etiquetas como forma de sinalizar à respectiva área ou tema da publicação para que ela seja encontrada de modo simples, rápido e estruturado. É solicitado ao usuário que este adicione no mínimo uma etiqueta, podendo chegar à quantidade máxima de cinco para representar a área do conteúdo abordado.

A relevância da plataforma para a área de desenvolvimento é evidente, uma vez que todos os conteúdos criados ficam disponíveis para a comunidade, muitas vezes facilitando e contribuindo para que futuras dúvidas sejam sanadas através da visualização de publicações anteriores [11].

A plataforma utiliza uma estratégia de gamificação para engajar os usuários de modo que os incentive a produzir e manter conteúdos de qualidade através de pontuações. O consumidor da publicação pode avaliar cada pergunta e comentário por meio de um sistema de votos, o qual funciona da seguinte maneira: o usuário ao avaliar uma pergunta ou resposta, recebe pontos de reputação, pontos esses que premiam o usuário com distintivos e emblemas. Quanto mais níveis o usuário subir, maior é seu privilégio dentro da comunidade, podendo ter até mesmo acesso a outras ferramentas.

Diversos estudiosos já utilizaram o *Stack Overflow* para fundamentar suas pesquisas e análises pelos mais variados segmentos da área do desenvolvimento de *software*. Diferentes autores resumiram as perguntas relacionadas ao desenvolvimento móvel na plataforma [12]; utilizaram o SO para entender os desafios e as necessidades dos desenvolvedores de *blockchain* [13]; e fizeram uso da modelação de tópicos para extrair tópicos de *big data* e os

---

[9] Plataforma do *Stack Overflow*. Disponível em: https://stackoverflow.com. Acesso em: 06 de outubro de 2022.

interesses dos desenvolvedores pelo assunto [14]. Logo, se faz notório os benefícios que ela traz para comunidade acadêmica.

Assim, devido a tamanha relevância e quantidade de informações presentes no *Stack Overflow*, podemos considerá-lo uma fonte de dados bastante pertinente. Entretanto, é humanamente impossível acessar e extrair informações das 23.091.888[10] perguntas disponíveis, até a presente data, de uma maneira manual. Utilizar técnicas de mineração de texto podem ser uma opção atrativa na extração e análise dessas informações.

2.2.1 *Modelagem de tópicos*

Modelagem de tópicos é uma forma de realizar a mineração de textos, que consiste em identificar padrões e realizar a extração de informações textuais, de modo que as intervenções humanas nesse processo sejam automatizadas.

A mineração de texto faz uso de diversas técnicas já utilizadas em uma mineração avançada de dados, como aprendizado de máquinas, recuperação de informações, linguagem computacional, como também no processamento de linguagem natural [15]. É explícita a relevância do uso destas técnicas para a extração de informações de texto, uma vez que se torna árdua a tarefa de manipulação de dados não estruturados.

Partindo desse ponto, foi desenvolvida a modelagem probabilística de tópicos, a qual se define como um conjunto de algoritmos que visam atrair informações sobre determinado tema em grandes arquivos de texto [16]. Estes algoritmos funcionam através de métodos focados no levantamento de estatísticas as quais investigam as palavras dos textos originários com o intuito de desvendar quais os temas abordados por meio de uma padronização das palavras encontradas, como eles se interligam e mudam ao longo do tempo [6].

Essa padronização se dá pela frequência que as palavras são usadas e pela simultaneidade destas mesmas frequências nos textos, para construir um modelo de palavras relativas que serão posteriormente resumidas em tópicos; e sendo possível classificá-los dentro de um conjunto [12].

Para inspecionar quantidades massivas de perguntas, é comum se fazer uso de abordagem que se baseia na *Latent Dirichlet Allocation* (LDA), um algoritmo de modelagem de tópicos [10]. Esse conjunto de operações é responsável por resumir grandes quantidades de arquivos textuais pelo método de aprendizagem não supervisionado [17].

Ao utilizar o LDA, um dos principais desafios é encontrar uma quantidade apropriada de tópicos para a análise, uma vez que a quantidade destes pode impactar na granularidade dos resultados, seja produzindo uma resposta bem específica ou produzindo uma mais genérica [18].

O método de LDA é amplamente utilizado em pesquisas focadas na modelação de tópico, sendo possível encontrar diversos estudos onde é aplicado para obtenção de dados do *Stack Overflow* [18]: foi realizado um estudo em grande escala no *Stack Overflow* e *GitHub*[11] utilizando LDA para encontrar assuntos debatidos por desenvolvedores em relação a três *frameworks* de *deep learning*: *Tensorflow*, *PyTorch* e *Theano* [19], assim como foi utilizado LDA no SO para identificar os interesses e desafios enfrentados por desenvolvedores de sistemas concorrentes [20].

---

[10] Número total de perguntas de acordo com a plataforma do Stack Overflow. Disponível em: https://stackoverflow.com/questions. Acesso em: 11 de outubro de 2022.

[11] Plataforma do GitHub. Disponível em: https://github.com/. Acesso em: 06 de outubro de 2022.

3 METODOLOGIA

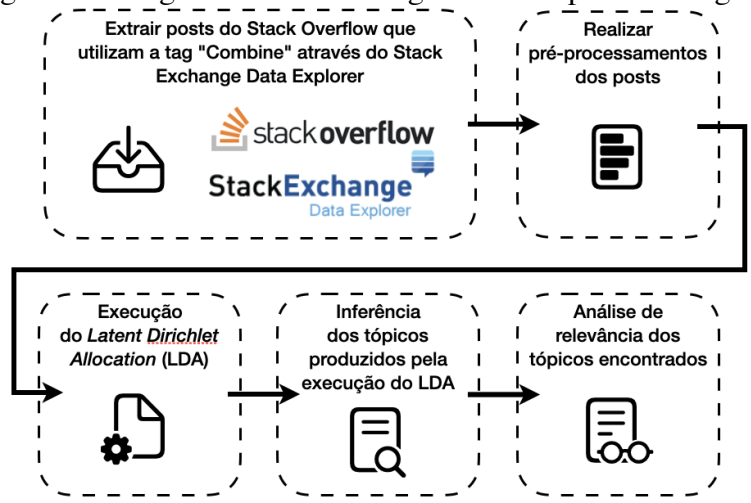
O principal objetivo deste estudo é entender quais os principais problemas e as maiores dúvidas enfrentadas por desenvolvedores na utilização do *Combine*. Para atingir o propósito foi utilizada a mineração de perguntas e respostas do *Stack Overflow* para obter os dados e informações necessárias para identificar essas barreiras.

O uso de mineração no *Stack Overflow* para análises e estudos similares já é utilizado em diversas áreas e domínios, como no caso do seu uso em um estudo da linguagem *Swift* [17], ou sobre a temática de concorrência [20], ou mesmo em estudos sobre programação reativa [18].

Nessa seção será descrito o processo de coleta de dados utilizando a mineração no *Stack Overflow*, bem como os passos realizados e as decisões tomadas durante o processo. Foi utilizado como base para a metodologia o próprio trabalho realizado por Zimmerle et al. [18], que apesar de possuir outros objetos de estudos, realiza os procedimentos necessários para obtenção dos dados para a realização da análise dessa pesquisa. Dessa forma, a linguagem utilizada para a mineração foi a linguagem *Go* em razão da utilização dos scripts produzidos pelo estudo [18].

De maneira geral, podemos resumir as etapas e o fluxo desse estudo conforme é exibido pelo diagrama da Figura 1.

Figura 1 – Diagrama com a visão geral das etapas e fluxo geral.



Fonte: da autora, 2022.

3.1 Mineração do Stack Overflow

Podemos dividir as etapas de mineração do *Stack Overflow* em cinco grandes etapas que são detalhadas a seguir: extração dos dados (Seção 3.1.1); pré-processamento dos dados (Seção 3.1.2); modelagem de tópicos com LDA (Seção 3.1.3); inferência dos tópicos (Seção 3.1.4) e análise de relevância dos tópicos (Seção 3.1.5).

3.1.1 *Extração dos dados*

Para realizar a mineração, o primeiro passo foi extrair os dados do *Stack Overflow* por meio do *Stack Exchange Data Explorer*. Com ela, é possível ter acesso atualizado aos dados

do SO e que podem ser baixados como um arquivo CSV através da execução de consultas SQL em uma interface *web*.

Para esse estudo, realizamos a extração das perguntas com a *tag* (ou etiqueta), '*combine*' e suas respectivas respostas aceitas, quando essas eram existentes, para termos o máximo de dados possíveis.

As perguntas e respostas são ambas consideradas como postagens, ou publicações, e esse conjunto de dados obtido totalizou um número de 2729 registros, com datas que variam desde 05 de Junho de 2019 (data após o lançamento do *framework Combine*) até 23 de Setembro de 2022 (data próxima ao dia que a consulta SQL foi realizada para esse estudo).

As informações extraídas de cada postagem vão desde o número identificador, título da pergunta, corpo do texto, data de criação, data da última atividade, data da última edição, pontuação, número de visualizações, entre outros.

3.1.2 *Pré-processamento dos dados*

A etapa de pré-processamento dos dados tem como objetivo tratar as postagens para que estejam prontas para a execução do LDA.

Para compor efetivamente o texto da postagem, quando esta é um a pergunta, foi considerada a combinação do seu título e do seu corpo do texto. Já para respostas aceitas, foi considerado o título da respectiva pergunta da resposta e o corpo de texto da resposta como composição ao corpo de texto da postagem.

Dessa forma, ao trabalhar com esses conjuntos de título e corpo de texto é possível aumentar a possibilidade de extrair também pedaços de código e mais outras informações relevantes da publicação que contribuem para as próximas etapas.

Após a composição para o texto da postagem, uma variedade de filtros é normalmente executada nos documentos. Funcionam como limpezas para aumentar a efetividade da execução do LDA. Assim, neste estudos, as seguintes remoções foram feitas: remoção de trechos de código; remoção de *tags* HTML; remoção de pontuação, caracteres não alfanuméricos, quebras de linha e sequência de espaços em branco; remoção de *stopwords*, palavras que não possuem relevância ao conteúdo e ao modelo; e remoção de números.

Além disso, depois de serem realizadas as remoções, também foi feita a stemmização, um processo que consiste em reduzir palavras flexionadas para o seu radical.

Outro tratamento realizado foi a remoção de palavras comuns, sendo essas as palavras que apareciam em um percentual de mais de 90% dos posts, e como também a remoção das seguintes palavras: '*answer*', '*question*', '*help'*, e '*solut'*. Também foram removidas as palavras incomuns, ou seja, aquelas que apareceram em até no máximo cinco postagens.

3.1.3 *Modelagem de tópicos com LDA*

Com as publicações limpas, chega o momento de executar o LDA. Essa etapa ocorreu conforme realizado em [18], onde foram mantidos os mesmos hiperparâmetros $\alpha = \beta = 0.01$ por se mostrarem mais coerentes para textos do *GitHub* e *Stack Overflow* [18]. Diante disso, como resultado foram obtidas variações de 10 até 35 tópicos, que foram manualmente inspecionados.

Tendo em vista a dificuldade de se determinar o valor ideal para o número de tópicos, a métrica de perplexidade também foi levada em consideração. Essa métrica ajudou a dar alguns palpites na hora de escolher a melhor quantidade de tópicos, uma vez que, ao diminuí-los a métrica tende a crescer [21]. Assim, quanto menor o valor da perplexidade, maior a chance de ser encontrado um modelo melhor.

Porém, a perplexidade e o julgamento humano nem sempre se correlacionam [25]. Então, para os diferentes resultados foram observados os valores para os quais os número de tópicos apresentavam pequenas melhorias na perplexidade. A série de tentativas e várias inspeções em resultados diversos foram determinantes para definir o número ideal de tópicos, sendo 20 a quantia mais coerente para esse estudo.

3.1.4 *Inferência dos tópicos*

A partir da execução do LDA, foram extraídos os tópicos. Cada um deles é composto por um conjunto de 20 palavras mais relevantes e 15 postagens obtidas de maneira aleatória. Sendo necessário a inferência do tópico ou rótulo por quem está realizando o estudo.

Para realizar essa inferência foi utilizado a técnica de classificação aberta de cartões, que consiste em analisar as palavras e as postagens aleatórias para cada tópico dominante. O estudo contou com a colaboração de outros dois especialistas no *framework Combine* para a execução da técnica.

A autora do estudo e a primeira especialista ficaram encarregadas de rotular os tópicos individualmente, aplicando a técnica de classificação aberta de cartões e, após essa primeira rodada de inferências, discutiram conjuntamente os resultados encontrados com um apoio de um mediador, o segundo especialista, que serviu para chegar em um acordo quando se era necessário.

Dessa forma, após esse momento de mediação foi possível definir os rótulos dos tópicos de maneira que estivessem em acordo com a autora e os dois especialistas.

3.1.5 *Análise de relevância dos tópicos*

Para realizar a análise de relevância dos tópicos é possível utilizar distintas métricas; nesse estudo foi optado por explorar as mesmas utilizadas que em [18]. Dessa maneira, as métricas utilizadas foram baseadas em popularidade e dificuldade.

A métrica de popularidade é calculada com base nas médias de três informações obtidas ainda na extração dos dados das postagens, sendo elas: o número de visualizações, ou seja, a quantidade de vezes que usuário do SO acessaram aquela publicação; o número de marcações como favorito; e a pontuação da postagem, valor esse obtido conforme a diferença entre os *upvotes* e *downvotes*.[12] Sendo ordenados primeiro pelo maior número médio de visualizações, seguido pelo maior número médio de marcações como favorito e por último pelo maior número médio da pontuação das postagens. Assim, quando um tópico apresenta um alto número de visualizações das suas postagens, um alto número de marcações como favoritos e uma alta pontuação, podemos compreender que aquele tópico é popular.

Para a métrica de dificuldade de um tópico, o estudo considerou o percentual de perguntas sem respostas aceitas e a média de tempo levado para que as respostas das perguntas fossem aceitas. Essa média de tempo é calculada com base em outras duas informações que também foram obtidas na hora da extração dos dados das postagens, sendo elas: as datas de criação das respostas aceitas e as datas de criação das respectivas perguntas. Dessa maneira, tópicos que apresentam um alto número de perguntas sem resposta, e que levaram mais tempo para terem respostas aceitas, são evidentemente mais difíceis.

---

[12] Análise de pontos realizada pela plataforma do Stack Overflow. Disponível em: https://stackoverflow.com/help/privileges/established-user. Acesso em: 07 de outubro de 2022.

4 RESULTADOS

Nesta seção são apresentados e discutidos os principais resultados obtidos da análise realizada pelo estudo, a fim de responder à questão de pesquisa proposta no presente artigo.

4.1 Tópicos inferidos

Após a execução da modelagem de tópicos e inferência dos mesmos, temos como resultado o que é visto no Quadro 1. Nele, é exibida todas as classificações dos 20 tópicos e suas palavras-chave associadas.

Podemos notar a partir desse resultado que as postagens no *Stack Overflow* sobre *Combine* estão associadas a uma ampla variedade de temas, onde os tópicos gerados correspondem a aspectos realmente relevantes e associados ao uso do framework.

Em geral, foi descoberto que as publicações acabam se enquadrando nas principais categorias de: abstrações da *stream*; concorrência; desenvolvimento de aplicações; erros semânticos; gerenciamento de estado; interface de usuário; *networking*; padrões arquiteturais; e programação.

Quadro 1 – Os 20 tópicos extraídos pelo LDA

Tópico	**Palavras**
Erros ligados a versionamento	*test io xcode combin app version code project bug beta error crash build fail develop devic simul wrong target messag*
Manipulação de dados obtidos de um URL	*api data combin json decod swift fetch url imag code struct return download error pars key object fail respons string*
Gerenciamento de memória de *streams*	*subscript cancel sink store combin set memori creat anycancel swift return retain publish leak object captur method complet dealloc code*
Gerenciamento de estado	*updat swiftui variabl bind initi view class pass code ui set publish app observableobject correct struct valu creat simpl expect*
Operações assíncronas	*user method check learn complet handler log program progress reactiv asynchron app start initi login callback tutori sign code global*
Manipulação de *array*	*array data object item list element core collect save context creat addit code delet updat model entiti queri pass databas*
Uso de *ViewModel* (MVVM)	*list swiftui load view app user updat appreci display data mvvm navig dynam item initi viewmodel build code screen firestor*
Uso de operadores	*combin publish emit oper valu multipl output swift creat framework stream upstream element produc combinelatest merg expect singl rxswift collect*
Uso de *publisher-subscriber*	*subscrib publish combin receiv document swift code appl pattern class write method implement creat understand post manag notif simpl deleg*
Interações na UI	*button tap control view uikit combin press action text code click enabl swiftui row idea track cell viewmodel viewcontrol trigger*
Requisições API REST	*request network combin respons code handl api error success return call swift server data publish token status result alamofir http*

Tipos de dados	*type option paramet generic conform extens protocol compil argument error unwrap syntax implement declar write question infer defin swift creat*
Erros de conversão de tipos	*error return publish combin type function swift convert method map chain futur closur anypublish result failur flatmap creat code oper*
Desenvolvimento com *SwiftUI*	*view swiftui parent viewmodel child init behavior bind code screen anim propag root dismiss depend subview question compon disappear contain*
Problemas genéricos	*code print execut publish playground timer combin block fire trigger start time function stop swift behaviour statement expect question correct*
Concorrência	*pipelin thread main background asynchron oper queue async task combin schedul complet process receiv perform run dispatch happen debounc synchron*
Data flow em *SwiftUI*	*view swiftui model object updat properti publish class observableobject pass creat instanc observ viewmodel data access observedobject environmentobject var environ*
Data Binding	*properti observ combin publish assign swift wrapper updat comput code valu swiftui set framework question trigger notifi notif access wrap*
Adaptação ao paradigma reativo	*send complet event publish finish combin subject passthroughsubject valu wait sequenc subscrib process pass oper previous delay receiv sink output*
Manipulação de dados de controles de entrada	*string textfield field input user text swiftui combin valid search type enter video code form creat charact wwdc locat idea*

Fonte: da autora, 2022.

Detalhando brevemente os tópicos identificados e seguindo a ordem exibida no Quadro 1, temos:

a) **Erros ligados ao versionamento** trata de publicações as quais desenvolvedores apresentam problemas ou dificuldades associados a erros de versão do ambiente ou de *target* de desenvolvimento;
b) **Manipulação de dados de um URL** está diretamente relacionado ao *download* e tratamento de mídias e outros dados obtidos a partir de um URL;
c) **Gerenciamento de memória de *streams*** trata de problemas relacionados a manter *publisher* ou *subscribers* alocados em memória de uma maneira correta ou eficiente;
d) O **gerenciamento de estado** envolve publicações relativas ao compartilhamento adequado dos dados de estado entre os componentes, nas quais é possível, por exemplo, identificar várias dúvidas quanto ao uso dos *property wrappers*;
e) O tópico de **operações assíncronas** é sobre dúvidas relativas ao entendimento e uso dessas operações, como por exemplo: checar o progresso de uma *task* da questão 62627660[13];
f) **Manipulação de *Array*** trata de publicações relativas a manipulação de *arrays* e coleções, como pode ser visto na questão 59879487[14];

[13] Pergunta "*Receive task progress from "URLSession.dataTaskPublisher*". Disponível em: https://stackoverflow.com/questions/62627660. Acesso em: 11 de outubro de 2022.

[14] Pergunta "Swift Combine - prefix publisher on array". Disponível em: https://stackoverflow.com/questions/59879487. Acesso em: 11 de outubro de 2022.

g) **Uso de *ViewModel* (MVVM)** se refere a dúvidas de implementação e entendimento no uso de *ViewModels* ao se utilizar o *Combine*, seguindo o padrão arquitetural MVVM, como exemplo, podemos citar a pergunta 64151269[15];
h) No tópico referente ao **uso de operadores** estão perguntas mais gerais quanto ao uso destes. Essas dúvidas vão desde simples perguntas relacionadas ao funcionamento de alguns operadores, como também sugestões de uso de outros, ou mesmo na identificação de operadores similares em outras soluções, como *RxSwift*;
i) **Uso de *publisher-subscriber*** são relacionados às publicações com dúvidas acerca do uso, comportamento ou funcionamento de *publishers* ou *subscribers*, ou mesmo questionamentos sobre a implementação de *publishers* personalizados;
j) **Interações na UI** são publicações referentes ao uso de *Combine* associado a interações na interface do usuário e seus elementos;
k) **Requisições API REST** estão associadas a perguntas e respostas para a realização de requisições HTTP e tratamento de suas respostas;
l) **Tipos de dados** são perguntas ligadas a dúvidas sobre os tipos utilizados pelos *Combine* e sua sintaxe;
m) **Erros de conversão de tipos** são questionamentos decorrentes do encadeamento de operadores que resultam em tipos genéricos aninhados os quais podem divergir do esperado para o desenvolvedor;
n) **Desenvolvimento com *SwiftUI*** estão relacionadas a postagens sobre o uso do *SwiftUI* no desenvolvimento de interfaces;
o) **Problemas genéricos** estão associados aos mais diversos tipos de perguntas envolvendo o *framework Combine*. Estas vão desde o diferente uso entre classe e estrutura, como o uso de timer ou até mesmo problemas de correção;
p) **Concorrência** trata de dúvidas referentes a operações assíncronas, seus mecanismos, ou mesmo o uso associado de threads;
q) ***Data flow* em *SwiftUI*** refere-se a perguntas associadas ao fluxo de dados sincronizados entre as camadas da aplicação e a *view* em *SwiftUI*;
r) ***Data Binding*** refere a questionamentos associados a ligação entre dados, como pode ser visto na pergunta 58810366[16];
s) **Adaptação ao paradigma reativo** trata de perguntas associadas a mudança de paradigma, quando esses estão tentando adaptar de um contexto imperativo para o uso da programação reativa pelo *Combine*;
t) E por fim as dúvidas ligadas à **manipulação de dados de controle de entrada** correspondem a questionamentos acerca da manipulação de informações oriundas do usuário através da interface, como por exemplo, a validação de um texto para a ativação de botão, como é possível identificar na pergunta 56910228[17].

Abaixo, expandimos nossas descobertas relacionadas aos tópicos encontrados, identificando quais são os mais populares e os mais difíceis dentre eles, além da realização de uma breve comparação com alguns trabalhos relevantes sobre programação reativa.

---

[15] Pergunta "*SwiftUI + Combine, using Models and ViewModels together*". Disponível em: https://stackoverflow.com/questions/64151269. Acesso em: 11 de outubro de 2022.

[16] Pergunta "*Two-way binding in Swift Combine*". Disponível em: https://stackoverflow.com/questions/58810366 . Acesso em: 11 de outubro de 2022.

[17] Pergunta "*SwiftUI Combine Debounce TextField*". Disponível em: https://stackoverflow.com/questions/66165075 . Acesso em: 11 de outubro de 2022.

4.1.1 *Tópicos identificados*

Ordenando o resultado obtido com relação à quantidade de postagens de cada tópico e sua respectiva categoria, temos a Tabela 1. Com ela, podemos verificar todos os tópicos identificados no resultado, assim como o respectivo número de postagens associadas e o percentual em relação ao total.

Tabela 1 - Tópicos identificados

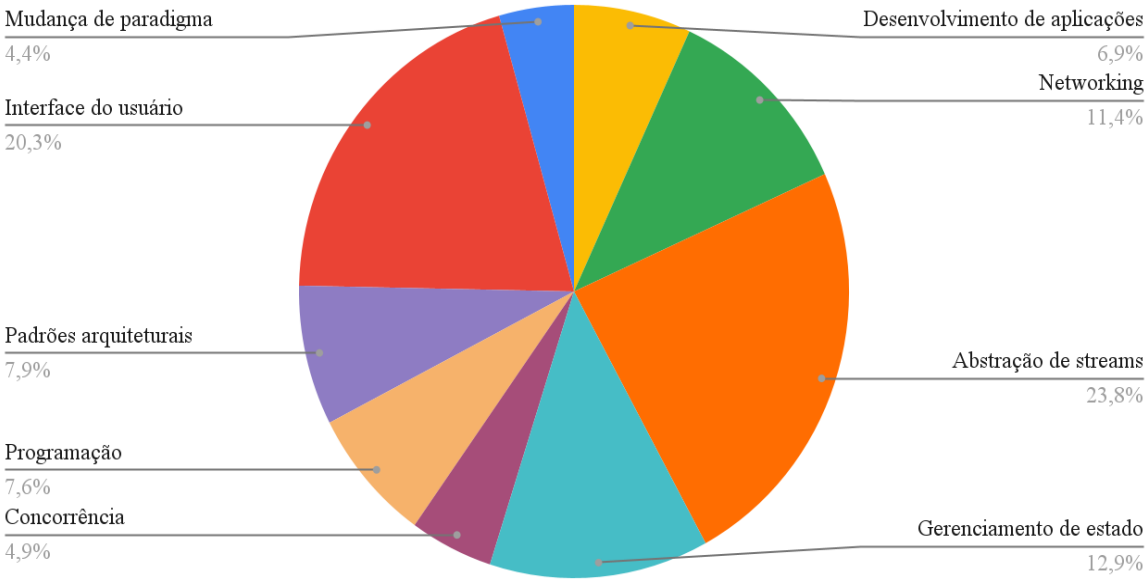
Tópico	**Categoria**	**Número de Postagens**	**% de Postagens**
Erros de conversão de tipos	Abstração de streams	320	11,7
Data flow em *Swift*UI	Interface do Usuário	302	11,1
Uso de operadores	Abstração de streams	207	7,6
Gerenciamento de estado	Gerenciamento de estado	184	6,7
Data Binding	Gerenciamento de estado	168	6,2
Requisições API REST	Networking	161	5,9
Manipulação de dados obtidos de um URL	Networking	149	5,5
Gerenciamento de memória de streams	Abstração de streams	124	4,5
Uso de publisher-subscriber	Padrões arquiteturais	123	4,5
Adaptação ao paradigma reativo	Mudança de paradigma	119	4,4
Manipulação de array	Programação	116	4,3
Problemas genéricos	Desenvolvimento de aplicações	98	3,6
Desenvolvimento com *Swift*UI	Interface do Usuário	93	3,4
Uso de ViewModel (MVVM)	Padrões arquiteturais	92	3,4
Erros ligados a versionamento	Desenvolvimento de aplicações	91	3,3
Manipulação de dados de controles de entrada	Interface do Usuário	90	3,3
Tipos de dados	Programação	90	3,3
Concorrência	Concorrência	76	2,8
Interações na UI	Interface do Usuário	69	2,5
Operações assíncronas	Concorrência	57	2,1
Número de postagens = 2.729			

Fonte: da autora, 2022.

Com relação à frequência de dúvidas, temos o tópico com o maior número de postagens sobre erros de conversão de tipos do *Combine*. Sabendo que a definição dos tipos é estabelecida em ambas as extremidades da pipeline, ou seja, pelo *publisher* em sua origem e pelo *subscriber* que consome o dado, ao compor uma *pipeline*, é preciso ter em mente que o encadeamento de operadores e outras funções resulta em tipos genéricos aninhados que podem divergir do que se é esperado, tornando em algo complexo para o desenvolvedor lidar. Como exemplo de uma pergunta pertencente a esse tópico, tempo a 60550577[18], onde o desenvolvedor deseja converter "...*Just<[Int]>*" para "*AnyPublisher<[Int], Error>*".

Já a Figura 2 exibe o percentual da quantidade de postagens de cada categoria. Nela podemos notar que a categoria com o maior número está relacionada aos tópicos de abstrações da *streams*, com 28% do total, e que compreende tópicos pertinentes a estruturação dos fluxos de dados numa lógica reativa. O que se mostra relevante ao confrontarmos com os dados de outro estudo, uma vez que essa categoria também representa a maior parcela dos tópicos identificados em Zimmerle et al [18].

Figura 2 –Percentual das postagens do Stack Overflow por categoria



Fonte: da autora, 2022.

A segunda categoria com maior quantidade de publicações é sobre tópicos relacionados à interface do usuário. Tendo em vista que o *Combine* é a implementação reativa para aplicações móveis que demandam uma interação com o usuário através de uma interface gráfica, acaba sendo evidente sua relação.

E não apenas isso, mas foi lançado juntamente com o *SwiftUI*, um *framework* declarativo para a construção de interfaces de usuários de dispositivos da plataforma Apple [22]. Ambos acabam comumente se relacionando, já que compartilham de alguns princípios e tanto as classes, como as estruturas utilizadas pelo *SwiftUI*, podem ser transformadas em *Publishers* ou *Subscribers* do *Combine* [23], o que o torna um companheiro lógico eficiente para seu uso.

---

[18] Pergunta "*Combine convert Just to AnyPublisher*". Disponível em: https://stackoverflow.com/questions/60550577. Acesso em: 11 de outubro de 2022.

A terceira categoria com a maior quantidade de publicações é a respeito do gerenciamento de estado; um conceito bastante relevante dentro da programação reativa [2] que obteve 12,9% do total das publicações.

4.1.2 *Tópicos mais populares*

Para avaliar os tópicos por uma perspectiva de relevância, utilizamos a métrica de popularidade conforme mencionado anteriormente. Ordenamos então os posts com base na quantidade de visualizações, a média de favoritos e a pontuação. Esse resultado pode ser visto na Tabela 2.

Tabela 2 – Tópicos mais populares

Tópico	**Visualizações**	**Média de Favoritos**	**Média de Pontuação**
Gerenciamento de estado	1961,4	0,5	3,10
Gerenciamento de memória de *streams*	1895,1	0,8	3,80
Concorrência	1883,2	1,6	4,80
Data flow em SwiftUI	1874,4	0,8	3,20
Data Binding	1752,8	0,8	3,40
Tipos de dados	1668,3	0,8	4,50
Manipulação de dados de controles de entrada	1640,4	0,7	2,20
Erros de conversão de tipos	1542,1	0,4	2,20
Uso de *publisher-subscriber*	1449,6	0,5	2,50
Uso de operadores	1195,6	0,3	1,80
Erros ligados a versionamento	1063,9	0,6	2,80
Adaptação ao paradigma reativo	1053,0	0,4	1,80
Desenvolvimento com *SwiftUI*	1009,8	0,7	2,00
Requisições API REST	974,4	0,3	1,20
Manipulação de *array*	956,5	0,8	1,60
Interações na UI	930,9	0,4	1,80
Uso de *ViewModel* (MVVM)	910,9	0,4	1,30
Manipulação de dados obtidos de um URL	862,4	0,2	0,70
Problemas genéricos	787,5	0,1	0,90
Operações assíncronas	740,6	0,3	1,30

Fonte: da autora, 2022.

Ao analisar os dados da tabela, podemos verificar que os três tópicos mais populares são sobre: gerenciamento de estado; gerenciamento de memória de *stream*; e concorrência. Ter o gerenciamento de estado como tópico de maior número de visualizações provavelmente se deve à necessidade de entendimento ao fato da inversão de controle que ocorre no paradigma que difere das estruturas imperativas. Como pode ser observado na pergunta 70569603[19], onde o usuário pede ajuda para conseguir sincronizar a atualização de todos os itens *Published* em um *ObservableObject*.

Já sobre o gerenciamento de memória de *stream*, podemos supor que ocorre devido à falta de entendimento em como se é gerenciado o ciclo de vida das *streams*. Como podemos verificar na pergunta 62786278[20], a qual o usuário se questiona como uma *subscription* não é desalocada em uma *view* modificada.

Sobre o tópico relacionado à concorrência, conseguimos enxergar o dado de forma intrigante, uma vez que nesta etapa da análise ele se apresenta como o tópico favorito dentre os outros e um dos que possui mais visualizações, porém, no exame seguinte, veremos que é o que apresenta maior dificuldade, tanto de compreensão, como de resolução.

4.1.3 *Tópicos mais difíceis*

Após classificar a complexidade e frequência dos tópicos, chegamos aos que são identificados como mais difíceis. Ordenando o resultado obtido com relação a quantidade de respostas não aceitas e o tempo médio de resposta para cada tópico, formamos a Tabela 3.

Através dela conseguimos interpretar que os assuntos com menos respostas aceitas são os que possuem mais complexibilidade e possível falha no acesso à informação, uma vez que o conhecimento do *framework* ainda está sendo construído de forma massiva.

Tabela 3 – Tópicos mais difíceis

Tópico	**Sem Respostas Aceitas (%)**	**Tempo Médio (hr)**
Concorrência	56,8	3,8
Uso de *ViewModel* (MVVM)	56,6	1,8
Desenvolvimento com *SwiftUI*	50,0	1,4
Interações na UI	47,5	5,4
Problemas genéricos	47,2	1,6
Uso de *publisher-subscriber*	46,8	3,1
Erros ligados a versionamento	46,7	7,5
Manipulação de dados obtidos de um URL	45,7	1,6
Data Binding	45,0	4,3
Manipulação de *array*	44,7	3,0

[19] Pergunta “Synchronise the update of all @Published items in an ObservableObject at the same time”. Disponível em: https://stackoverflow.com/questions/70569603 . Acesso em: 10 de outubro de 2022.

[20] Pergunta “Why *does this combine subscription not deallocate in custom ViewModifier*?” Disponível em: https://stackoverflow.com/questions/62786278 . Acesso em: 10 de outubro de 2022.

Requisições API REST	43,9	5,9
Gerenciamento de memória de streams	43,1	2,6
Operações assíncronas	40,9	1,6
Adaptação ao paradigma reativo	40,8	2,1
Uso de operadores	40,7	1,7
Data flow em *SwiftUI*	40,2	1,1
Erros de conversão de tipos	37,0	1,9
Gerenciamento de estado	34,7	1,5
Manipulação de dados de controles de entrada	29,9	2,2
Tipos de dados	21,4	1,1

Fonte: da autora, 2022.

Para o tópico de concorrência, mesmo que o seu tempo médio de resposta seja menor do que alguns que possuem mais respostas aceitas, ele termina sendo classificado como o que possui mais dúvidas por essa menor quantidade de respostas aceitas, talvez justamente pela complexidade natural do tema.

Situação oposta às perguntas sobre os problemas com uso de *ViewModel* no *Combine*, uma vez que mesmo sem muitas respostas aceitas, o tempo de resposta para as postagem sobre o assunto chega a ser quase a metade quando comparado ao de concorrência. Assim, podemos afirmar que, em termos de acesso ou interesse, ele ainda é um tema mais usual aos desenvolvedores do que o tópico de concorrência. Podendo ser isso uma consequência direta da jornada de aprendizado de um desenvolvedor *Swift*, ao qual tópicos relacionados à arquitetura acabam por vezes sendo mais abordados do que tópicos de concorrência.

5 CONCLUSÃO

Sendo um *framework* relativamente novo, já que foi lançado em 2019, até onde se sabe, existe pouca evidência de pesquisas relacionadas ao seu uso e nenhuma que tenha estudado postagens relacionadas a *Combine* usando o *Stack Overflow*.

Da mesma maneira que existe uma necessidade de mais estudos alinhados com a capacidade de aprendizado e compreensão por novos desenvolvedores ou mesmo por aqueles que já possuem experiência na construção de aplicações móveis que utilizam *frameworks* de programação reativa. Acreditamos que esse estudo, ao analisar as postagens relacionadas ao *Combine* no SO, pode complementar com o trabalho que trata do uso de programação reativa com *ReactiveX* analisando postagens do *Stack Overflow* e repositórios do *GitHub* [18].

Extraímos os tópicos relacionados ao uso do *Combine* e os categorizamos. Além disso, examinamos a quantidade, a popularidade e a dificuldade desses tópicos. Acreditamos que esse trabalho abre portas para a comunidade de pesquisa iniciar investigações e sugerir aprimoramentos para tornar o uso do *framework* mais acessível para desenvolvedores de *Swift* interessados no uso de programação reativa. Da mesma forma que pode colaborar para a identificação de problemas comuns e mitigação desses quando o *framework* é utilizado.

Ademais, para complementar este trabalho seria relevante executar entrevistas com desenvolvedores que utilizam o *framework* para a realização de um *cross-validation* do que

foi encontrado no presente estudo e da perspectiva singular de cada programador, ou seja, confrontar os dados obtidos atualmente com outras fontes.

Se mostra relevante também um estudo voltado para a correlação entre as maiores dúvidas no *Stack Overflow* sobre *Combine* e o seu concorrente, o *RxSwift*, avaliando se as dúvidas são semelhantes e o nível de complexidade relacionado a cada ferramenta.

Outro tema que seria interessante em abordar é a análise se houve algum aumento ou diminuição no interesse pelas produções de aplicativos utilizando *RxSwift* desde o lançamento do Combine.

REFERÊNCIAS

[1] SALVANESCHI, G.; MARGARA, A.; TAMBURRELLI, G.. Reactive Programming: a walkthrough. *In:* IEEE International Conference on Software Engineering, 37., 2015, [S.l.]. **Proceedings [...].** p. 953-954. DOI 10.1109/icse.2015.303.

[2] BAINOMUGISHA, E.; CARRETON, A. L.; VAN CUTSEM, T.; MOSTINCKX, S.; MEUTER, W.. A survey on reactive programming. **Acm Computing Surveys**, [S.l.], v. 45, n. 4, p. 1-34, ago. 2013. DOI 10.1145/2501654.2501666.

[3] BONÉR, J; D. FARLEY, D; KUHN, R; THOMPSON; M. **The reactive manifesto.** 2014. Disponível em: https://www.reactivemanifesto.org/. Acesso em: 10 out. de 2022

[4] STUKALOV, I. S.; MAMEDOVA, N. A.; STAROVEROVA, O. V.; MAKARENKOVA, E. V.. Method for Organizing Network Requests in iOS Applications. *In:* International Siberian Conference On Control And Communications, 15., 2021, [S.l.], **Proceedings [...].** p. 1-9. DOI 10.1109/sibcon50419.2021.9438849.

[5] HOLOPAINEN, N. **Reactive iOS Development with Rx*Swift***. 2022. TCC (Graduação em Engenharia), Metropolia University Of Applied Sciences, [S.L.], 2022.

[6] BLEI, D. M.; NG, A. Y.; JORDAN, M. I.. Latent dirichlet allocation. **Journal of Machine Learning Research**, [S.l.], v. 3, p. 993–1022, jan. 2003.

[7] SALVANESCHI, G.; PROKSCH, S.; AMANN, S.; NADI, S.; MEZINI, M.. On the Positive Effect of Reactive Programming on Software Comprehension: an empirical study. **Ieee Transactions On Software Engineering**, [S.l.], v. 43, n. 12, p. 1125-1143, 1 dez. 2017. DOI 10.1109/tse.2017.2655524.

[8] APPLE. **Receiving and Handling Events with Combine**. Disponível em: https://developer.apple.com/documentation/combine/receiving-and-handling-events-with-combine. Acesso em: 10 out. 2022.

[9] APPLE. **Combine**: customize handling of asynchronous events by combining event-processing operators. Disponível em: https://developer.apple.com/documentation/combine. Acesso em: 10 out. 2022.

[10] BARUA, A.; THOMAS, S. W.; HASSAN, A. E.. What are developers talking about? An analysis of topics and trends in Stack Overflow. **Empirical Software Engineering**, [S.l.], v. 19, n. 3, p. 619-654, 1 nov. 2012. DOI 10.1007/s10664-012-9231-y.

[11] ROCHA, A. M. **Documentação automatizada de APIs com tutoriais gerados a partir do Stack Overflow**. 2016. Dissertação (Mestrado em Ciência da Computação). – Faculdade de Computação, Universidade Federal de Uberlândia, Uberlândia, 2016.

[12] ROSEN, C.; SHIHAB, E.. What are mobile developers asking about? A large scale study using stack overflow. **Empirical Software Engineering**, [S.L.], v. 21, n. 3, p. 1192-1223, 21 abr. 2015. DOI 10.1007/s10664-015-9379-3.

[13] WAN, Z.; HUDAK, P.. Functional reactive programming from first principles. *In:* Acm Sigplan 2000 Conference On Programming Language Design And Implementation, 2000, [S.l.]. **Proceedings [...].** 2000. p. 242-252. DOI 10.1145/349299.349331.

[14] BAGHERZADEH, M; KHATCHADOURIAN, R., Going Big: A Large-Scale Study on What Big Data Developers Ask. *In:* aCM Joint Meeting on European Software Engineering Conference and Symposium on the FounDations of Software Engineering, 27., 2019, [S.l.]. **Proceedings [...]**. [S.l.], Association for Computing Machinery, 2019. p. 423-442. DOI doi/10.1145/3338906.3338939.

[15] FALEIROS, Thiago de Paulo. **Propagação em grafos bipartidos para extração de tópicos em fluxo de documentos textuais**. 2016. Tese (Doutorado em Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2016. DOI 10.11606/T.55.2016.tde-10112016-105854.

[16] BIANCHINI, L.. **Análise Explicatória dos Tópicos no Stack Overflow Usando LDA (Latent Dirichlet Allocation)**. 2018. TCC (Graduação em Ciência da Computação) – Universidade da Fronteira Sul, Chapecó, 2018.

[17] REBOUÇAS, M.; PINTO, G.; EBERT, F.; TORRES, W.; SEREBRENIK, A.; CASTOR, F.. An Empirical Study on the Usage of the *Swift* Programming Language. *In:* International Conference on Softwares Analysis, Evolution, and Reengineering, 23., 2016, [S.l.]. **Proceedings [...]**. 2016, p. 634-638. DOI 10.1109/SANER.2016.66.

[18] ZIMMERLE, C.; GAMA, K.; CASTOR, F.; MOTA FILHO, J. M.. Mining the Usage of Reactive Programming APIs: A Study on GitHub and Stack Overflow. *In:* International Conference on Mining Software Repositories, 19., 2022, [S.l.]. **Proceedings [...]**. 2022, p. p. 203-214. DOI 10.1145/3524842.3527966.

[19] HAN, J.; SHIHAB, E.; WAN, Z.; DENG, S.; XIA, X.. What do Programmers Discuss about Deep Learning Frameworks. **Empirical Software Engineering**, [S.l.], v. 25, n. 4, p. 2694-2747, 24 abr. 2020. DOI 10.1007/s10664-020-09819-6.

[20] AHMED, S.; BAGHERZADEH, M.. What Do Concurrency Developers Ask About? A Large-scale Study Using Stack Overflow. *In:* International Symposium on Empirical Software Engineering and Measurement, 12., 2018, [S.l.]. **Proceedings [...]**. 2018, p. 1-10. DOI 10.1145/3239235.3239524.

[21] AGRAWAL, Amritanshu; FU, Wei; MENZIES, Tim. What is wrong with topic modeling? And how to fix it using search-based software engineering. **Information And Software Technology**, [S.l.], v. 98, p. 74-88, jun. 2018. DOI 10.1016/j.infsof.2018.02.005.

[22] APPLE. ***Swift*UI**: declare the user interface and behavior for your app on every platform. Disponível em: https://developer.apple.com/documentation/*swift*ui. Acesso em: 10 out. 2022.

[23] HECK, J. **Using Combine**: *Swift*UI and Combine. 2022. Disponível em: https://heckj.github.io/*swift*ui-notes/#*swift*ui-and-combine. Acesso em: 10 out. 2022.

[24] ABDELLATIF, A.; COSTA, A.; BADRAN, K.; ABDALKAREEM, R; SHIHAB, E.. Challenges in Chatbot Development: A Study of Stack Overflow Posts. *In:* International Conference on Mining Software Repositories, 17., 2020, Seoul. **Proceedings [...]**. Seoul, Association for Computing Machinery. 2020. DOI 10.1145/3379597.3387472.

[25] CHANG, J.; BOYD-GRABER, J.; GERRISH, S.; WANG, C.; BLEI, D. M.. Reading tea leaves: How humans interpret topic models. *In:* International Conference on Neural Information Processing Systems, 22., 2009, [S. l.], **Proceedings [...]**. p. 288–296. 2009.

[26] NASCIMENTO. I. D. R.. **CEPCombine**: Uma biblioteca para o Processamento de Eventos Complexos em *Swift*. 2019. TCC (Graduação em Ciência da Computação) – Centro de Informática, Universidade Federal de Pernambuco, Recife, 2019. Disponível em: https://www.cin.ufpe.br/~tg/2019-2/TG_CC/tg_idrn.pdf. Acesso em: 10 out. 2022.

[27] HECK, J. **Using Combine**: core concepts: publisher and subscriber. 2022. Disponível em: https://heckj.github.io/*swift*ui-notes/#coreconcepts-publisher-subscriber. Acesso em: 10 out. 2022.