UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

ESCOLA DE INFORMÁTICA APLICADA

CARACTERIZAÇÃO DAS DOCUMENTAÇÕES DO DESENVOLVEDOR EM FERRAMENTAS DE DESENVOLVIMENTO DE SOFTWARE: UM ESTUDO DE CASO

BEATRIZ BARRETO VIANNA E SILVA CORONHA

Orientador

PAULO SÉRGIO MEDEIROS DOS SANTOS

RIO DE JANEIRO, RJ - BRASIL

OUTUBRO DE 2021

Catalogação informatizada pelo(a) autor(a)

C822

Coronha, Beatriz Barreto Vianna e Silva
Caracterização das documentações do desenvolvedor em ferramentas de desenvolvimento de software: um estudo de caso / Beatriz Barreto Vianna e Silva Coronha. -- Rio de Janeiro, 2021.
59 f.

Orientador: Paulo Sérgio Medeiros dos Santos.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Estado do Rio de Janeiro, Graduação em Sistemas de Informação, 2021.

1. Documentação. 2. Desenvolvimento de Software. I. Santos, Paulo Sérgio Medeiros dos, orient. II. Título.

CARACTERIZAÇÃO DAS DOCUMENTAÇÕES DO DESENVOLVEDOR EM FERRAMENTAS DE DESENVOLVIMENTO DE SOFTWARE: UM ESTUDO DE CASO

BEATRIZ BARRETO VIANNA E SILVA CORONHA

Trabalho de Conclusão de Curso, apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação

Aprovado por:

_____

Prof. DSc. Paulo Sérgio Medeiros dos Santos (Orientador)

Universidade do Estado do Rio de Janeiro (UNIRIO)

_____

Prof. DSc. Jobson Luiz Massollar da Silva

Universidade do Estado do Rio de Janeiro (UNIRIO)

_____

Prof. MSc. Talita Vieira Ribeiro

Universidade Federal do Rio de Janeiro (UFRJ)

RIO DE JANEIRO, RJ - BRASIL

OUTUBRO DE 2021

CORONHA, Beatriz Barreto Vianna e Silva. **CARACTERIZAÇÃO DAS DOCUMENTAÇÕES DO DESENVOLVEDOR EM FERRAMENTAS DE DESENVOLVIMENTO DE SOFTWARE: UM ESTUDO DE CASO.** 2021; 59 f. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) - Escola de Informática Aplicada, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2021.

RESUMO

Contexto: A documentação do desenvolvedor tem um papel fundamental no aprendizado de tecnologias de desenvolvimento de software. Porém, há indícios de que as documentações oficiais das tecnologias não são adequadas, o que faz com que os desenvolvedores tenham que recorrer a outras formas de conseguir a informação de que precisam. **Objetivo:** Identificar como as documentações costumam ser estruturadas na indústria, quais tipos de conhecimentos são capturados e qual a percepção da comunidade de desenvolvedores sobre as documentações. **Método:** Um estudo de caso múltiplo holístico analisando documentações de seis ferramentas de desenvolvimento para responder às três questões de pesquisa. Os casos foram analisados em conjunto de forma comparativa. **Resultados:** Os principais resultados encontrados baseados nas questões de pesquisa propostas foram: (i) O tipo mais comum de conhecimento apresentado nas documentações descreve as funcionalidades da ferramenta e o que acontece quando ela é usada; (ii) a estrutura das documentações não apresenta ampla variação, com a maioria das analisadas com hierarquias de seções entre dois e três níveis, além de seções com mesmo nome entre diferentes ferramentas (ex: Tutorial e FAQ); e (iii) foi encontrada uma potencial correlação entre a variedade de tipos de conhecimento em uma documentação e sua facilidade de uso. **Conclusão:** De forma geral, as documentações têm proporção de tipos de conhecimento e estrutura similares, o que corrobora uma visão compartilhada entre literatura técnica e profissionais da indústria de que as deficiências atingem a maioria das documentações. Os resultados deste estudo contribuem para o entendimento das documentações do desenvolvedor e servem de base para um diagnóstico das documentações atuais. Esses resultados podem servir como ponto de partida para o estudo de formatos alternativos de documentação que possam se mostrar melhores que os atuais.

Palavras-chave: Documentação, Desenvolvimento de Software

CORONHA, Beatriz Barreto Vianna e Silva. **CHARACTERIZATION OF DEVELOPER DOCUMENTATIONS IN SOFTWARE DEVELOPMENT TOOLS:A CASE STUDY.** 2021; 59 f. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) - Escola de Informática Aplicada, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2021.

ABSTRACT

Context: Developer documentation plays a key role in learning software development technologies. However, there are indications that the official documentation of the technologies is not suitable, which makes developers have to resort to other ways to get the information they need. **Objective:** Identify how documentations are usually structured in the industry, what knowledge is captured and what the developer community's perception of documentation is. **Method:** A holistic multiple case study analyzing documentation from six development tools to answer three research questions. The cases were analyzed together in a comparative way. **Results:** The main results found based on the proposed research questions were: (i) The most common type of knowledge presented in the documentation describes the functionality of the tool and what happens when it is used; (ii) the structure of the documentation does not vary widely, with most of those analyzed with hierarchies of sections between two and three levels, in addition to sections with the same name between different tools (eg Tutorial and FAQ); and (iii) a potential correlation was found between the variety of knowledge types in a documentation and its ease of use. **Conclusion:** In general, documentation has a similar proportion of types of knowledge and structure, which corroborates a shared view between technical literature and industry professionals that deficiencies in documentation affect most of them. The results of this study contribute to the understanding of the developer's documentation and serve as a basis for a diagnosis of current documentation. In addition, they are expected to serve as a starting point for the study of alternative documentation formats that may prove to be better than current ones.

Keywords: Documentation, Software Development

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1. Introdução

Profissionais da área de desenvolvimento de software utilizam diversas tecnologias de desenvolvimento de software para criarem seus projetos, incluindo a linguagem de programação escolhida para o sistema, o banco de dados que será utilizado, frameworks de desenvolvimento ou bibliotecas de código. Porém, para que o desenvolvedor consiga utilizar essas tecnologias, é necessário que entenda como elas devem ser utilizadas e, para isso, precisará das documentações dessas tecnologias, tal como um manual descrevendo para que elas servem e como devem ser utilizadas. Essas documentações normalmente são disponibilizadas nos sites oficiais das próprias tecnologias ou, no caso de projetos de código aberto, no próprio repositório do sistema de controle de versões onde o projeto é disponibilizado.

Como o mercado de Tecnologia de Informação (TI) está em constante crescimento e mudança, isto faz com que o desenvolvedor tenha que continuamente aprender novas tecnologias de desenvolvimento ou suas evoluções. Um desafio neste processo é que essas tecnologias, além de estarem sempre sendo atualizadas, frequentemente possuem complexidade de uso elevada. Portanto, a busca pelo conhecimento faz parte das profissões nas áreas de tecnologia: “um dos desafios de ser desenvolvedor hoje é que habilidades de aprendizagem ao longo da vida são necessárias para desfrutar uma carreira de sucesso.”(BEGEL e SILLITO, 2013, tradução nossa)[1].

Uma das principais formas de aprender uma nova tecnologia de desenvolvimento de software é procurando sua documentação, quando não a única disponível. Porém, nem sempre essa documentação está clara o bastante para ser entendida ou possui informações úteis (ROBILLARD, 2009). Isto representa um problema para a utilização da tecnologia, pois esse deveria ser um dos principais pontos de contato entre o desenvolvedor e a tecnologia que ele está aprendendo ou utilizando. Tecnologias e serviços como StackOverflow[2] representam complementos importantes à documentação, mas não a substituem completamente (BALTES, 2020).

[1] No original: One of the challenges of being a developer today is that life-long learning skills are required to enjoy a successful career
[2] https://stackoverflow.com/

Diante deste diagnóstico de uma documentação com qualidade insuficiente, esse trabalho visa caracterizar a documentação atualmente disponibilizada em tecnologias de desenvolvimento de software. Com isto, imagina-se que futuros trabalhos possam entender o que faz com que uma documentação seja considerada útil, o que ela precisa ter e como ela deve ser estruturada.

1.1. Motivação

Para Meng et al (2020), a documentação apresenta um importante papel de reduzir problemas de usabilidade que não podem ser resolvidos por meio de decisões de design ou no apoio oferecido pela ferramenta. Esses problemas podem ocorrer, por exemplo, quando o desenvolvedor da ferramenta toma decisões no projeto que irão melhorar o desempenho, mas comprometem a usabilidade e o entendimento da tecnologia. Quando a documentação não cumpre seu papel de comunicar de forma clara, acaba se tornando uma frustração para o desenvolvedor, podendo fazer com que ele desista de usar determinada tecnologia.

Segundo uma pesquisa feita pela plataforma GitHub em 2017 (MOTROC, 2017), 72% dos desenvolvedores responderam que sempre buscam projetos open source quando estão avaliando novas tecnologias, porém 93% dos participantes reclamaram da documentação de projetos open source. Ainda assim, 60% dos desenvolvedores responderam que não se interessam em contribuir para melhorá-la.

Um indício de que as documentações atuais apresentam problemas é a popularização de fóruns de discussão, como o *StackOverflow*. Nesta plataforma, usuários compartilham suas dúvidas, conhecimentos e experiências com diferentes tecnologias. Segundo uma pesquisa feita por Parnin (2013), desenvolvedores podem obter até 50% da documentação de uma ferramenta no *StackOverflow*. De acordo com os resultados, os principais motivos para isso são que no *StackOverflow* é possível encontrar o dobro de exemplos do que nas documentações oficiais e as explicações são mais completas e mais fáceis de entender. A documentação oficial do Android, por exemplo, só tinha explicações para 36% de suas classes, enquanto que o *StackOverflow* apresentava explicações para 88%. No momento da escrita deste trabalho ele é o 45º site mais acessado no mundo[3].

Portanto, é possível perceber, pela popularização do *StackOverflow*, que os desenvolvedores têm necessidade de informações e conhecimento sobre as diferentes tecnologias. Porém, essa necessidade seria melhor atendida se as documentações oficiais

[3] https://www.alexa.com/topsites

fossem melhor elaboradas. Isso porque, por mais que seja possível encontrar informações no *StackOverflow*, não é possível encontrar todo tipo de resposta. Normalmente encontram-se pequenos exemplos e dúvidas menores sobre o uso das tecnologias. Explicações mais abrangentes sobre as funcionalidades, seu propósito e padrões de uso costumam aparecer somente nas documentações.

1.2. Objetivo

Esse trabalho visa caracterizar as documentações do desenvolvedor na indústria e diagnosticar suas limitações e deficiências. A partir desse objetivo foram definidas três questões de pesquisa:

Questão de Pesquisa 1 (QP1): Quais os tipos de conhecimento são capturados nas documentações do desenvolvedor?

Questão de Pesquisa 2 (QP2): Quais as estruturas de documentação do desenvolvedor para tecnologias de desenvolvimento de software são utilizadas?

Questão de Pesquisa 3 (QP3): Qual a percepção dos desenvolvedores sobre a qualidade das documentações?

1.3. Organização do Texto

O presente trabalho está estruturado em capítulos e, além desta introdução, está apresentado na seguinte estrutura:

O Capítulo 2 descreve os principais conceitos estudados.

O Capítulo 3 apresenta a metodologia usada neste estudo, incluindo como foi feita a seleção dos casos. A coleta de dados foi feita de forma manual avaliando as documentações selecionadas de forma parcialmente estruturada.

O Capítulo 4 analisa os resultados obtidos, respondendo às questões propostas. Além de discutir as implicações dos resultados.

Por fim, o Capítulo 5 reúne as considerações finais, limitações e deficiências encontradas e sugere possibilidades de aprofundamento futuro.

2. Fundamentação Teórica

Para realizar o estudo é necessário formalizar os conceitos fundamentais. Dessa forma, este capítulo apresenta esses conceitos com a seguinte organização: a subseção 2.1 define o que é uma tecnologia de desenvolvimento de software, a subseção 2.2 descreve a documentação do desenvolvedor, especificando suas características básicas. A subseção 2.3 apresenta trabalhos relacionados a esse estudo.

2.1. Tecnologia de Desenvolvimento de Software

Tecnologias de desenvolvimento de software representam métodos, técnicas ou ferramentas usadas no desenvolvimento de software. De acordo com Falbo, Menezes e Rocha (1998), métodos se referem a um procedimento sistemático, definindo etapas e heurísticas para a realização de uma ou mais atividades. Técnica é um procedimento para a realização de uma atividade, menos rígida e detalhada do que um método. Ferramentas são recursos de software utilizados para apoiar atividades, automatizando, parcialmente ou não, uma atividade. Recursos podem ser divididos em recursos de software, recursos de hardware e recursos humanos. Neste trabalho, o foco são os recursos de software.

Neste trabalho, o maior foco será nas ferramentas de desenvolvimento, as quais podem ter sido construídas em diferentes linguagens de programação e estarem disponíveis para diferentes plataformas ou sistemas operacionais. Documentação de métodos e técnicas não estarão no contexto deste trabalho.

De maneira geral, parte significativa das ferramentas de desenvolvimento são utilizadas por meio das Interfaces de Programação de Aplicações (APIs), como no caso de bibliotecas, *frameworks* e plataformas. Por conta disso, na literatura técnica, a maioria dos trabalhos trata do problema da documentação das APIs e não de ferramentas de desenvolvimento (ROBILLARD, 2009, ROBILLARD et al, 2017, MENG et al, 2020, THAYER et al, 2021). Neste contexto específico, consideramos APIs como um sinônimo para ferramentas de desenvolvimento de software.

“API são definidas como interfaces legíveis por máquina que conectam múltiplas aplicações e seus módulos, governam a interação de aplicativos e eliminam a necessidade de conhecer o comportamento interno de como a funcionalidade de uma API é fornecida”

(WULF e BLOHM, 2020, tradução nossa)[4]. APIs também podem ser divididas em diferentes categorias: as que estruturam a integração entre módulos de um mesmo sistema e as que fornecem acesso remoto pela Internet; as privadas, que só podem ser acessadas de dentro de uma rede específica e as públicas, que podem ser acessadas de fora dessas redes. As especificações das APIs normalmente são divulgadas abertamente para promovê-las para a comunidade de desenvolvedores.

2.2. Documentação do Desenvolvedor

Associada à maioria das atividades de engenharia de software está a documentação, mais especificamente a documentação do desenvolvedor. Esse tipo de documentação ajuda os desenvolvedores a aprender frameworks e bibliotecas (DAGENAIS e ROBILLARD, 2010). Os principais tipos de documentação do desenvolvedor são: comentários do código fonte, tutoriais e documentação de referência para as APIs. “Documentação do desenvolvedor é considerada uma das formas de informação mais úteis usada por desenvolvedores durante a manutenção de produtos de software” (LETHBRIDGE et al, 2003 apud ROBILLARD et al, 2017 tradução nossa)[5].

Para este trabalho, está sendo considerado “documentação” o conjunto de informações das ferramentas. Cada documentação pode conter um ou mais documentos que podem conter uma ou mais seções. Como estão sendo observadas ferramentas cujas documentações estão disponíveis em páginas na internet, os documentos são, normalmente, *links* separados entre as opções de acesso da documentação.

2.2.1. Tipos de documentação

A documentação compreende muitos tipos de artefatos que possuem diferentes finalidades, por isso, torna-se útil classificá-los para facilitar o entendimento. Uma classificação possível divide documentos em documentação interna e documentação externa.

Documentação interna consiste nas informações contidas no software que descreve o programa e sua evolução. Esse tipo de documentação descreve a estrutura de dados, algoritmos e o fluxo de dados do software (GAGGERO, 1983). “A documentação interna normalmente é elaborada para otimizar os seguintes atributos de qualidade do software:

[4] No original: We define APIs as machine-readable interfaces that connect multiple applications, govern application interaction, and remove the need to know the inner workings of how an API’s functionality is provided

[5] No original: Developer documentation is considered to be one of the most useful pieces of information by developers during software maintenance

mantenabilidade, adaptabilidade, portabilidade." (JUBENVILLE et al., 1976 apud GAGGERO, 1983 tradução nossa)[6]. Uma das principais formas de documentação interna são os comentários presentes no código do programa para torná-lo entendível sem depender de documentações externas ao código.

Documentação externa, por outro lado, foca na descrição do problema juntamente com o programa usado no apoio da sua solução, descrevendo a abordagem utilizada para tratar o problema. Esse tipo de documentação costuma acompanhar diagramas para mostrar como um componente se relaciona com outro. Segundo Gaggero (1983), a documentação externa pode ser dividida em três tipos: (i) documentação introdutória, direcionada ao usuário potencial do programa e que pode conter uma breve descrição do programa e de suas funcionalidades, (ii) documentação do usuário, destinada aos indivíduos que têm um interesse nas operações e aplicações do programa, contém manuais de uso e operação e guias de instalação, e (iii) documentação de manutenção, destinada aos desenvolvedores responsáveis por manter e modificar o programa, contém histórico de arquivos, manuais de teste e manual do sistema, que descreve todos os detalhes técnicos do programa.

De acordo com Abrahamson (2016), no caso da documentação de bibliotecas, pode haver confusão entre os tipos documentação (externa) de usuário e documentação interna, já que é comum escrever a documentação de bibliotecas dentro de um programa na forma de comentários, utilizando um software para extrair esses comentários e colocar em uma forma adequada para desenvolvedores que queiram usar a biblioteca. Um exemplo disso é a documentação do Java, feita por um software chamado Javadoc que lê um programa em Java, incluindo comentários, e escreve a documentação. Porém, a documentação de bibliotecas, por ser documentação externa, foca mais em descrever como os desenvolvedores podem utilizar as bibliotecas. Enquanto que a documentação interna está relacionada com o entendimento do programa e sua estrutura.

De acordo com Robillard e DeLine (2011), outra forma de classificar os tipos de documentação, e mais específica para documentação do desenvolvedor, é pensando nela como de alto ou baixo nível. Documentação de baixo nível é aquela que detalha de maneira mais técnica os aspectos da API, como códigos de erros e parâmetros de métodos. Por sua vez, a documentação de alto nível trata de conhecimentos mais conceituais, ou seja, quaisquer

[6] No original: Internal documentation should be developed to optimize the following software attributes: maintainability, adaptability, portability

conhecimentos necessários para utilizar uma ferramenta incluindo, por exemplo, definições do domínio ou boas práticas

Neste trabalho, o foco está na documentação externa (introdutória e do usuário) e de alto nível. É importante ressaltar que no contexto de ferramentas de desenvolvimento de software o "usuário" da documentação externa é o desenvolvedor. Por isso, a expressão documentação do desenvolvedor.

2.2.2. Características da documentação

Para que uma documentação seja útil, ela precisa atender determinadas condições. Sobre isso Thayer et al. (2021) afirmam que sua "teoria começa com a premissa de que usar uma API envolve identificar os requisitos para o comportamento desejado e, em seguida, utilizar a API para compor abstrações em programas que atendam a esses requisitos." Esse processo envolve três classes de conhecimento: conceitos de domínio, fatos de execução e padrões de uso de API.

Conceitos de domínio são noções abstratas ou do mundo real que existem no domínio da API e que ela tenta modelar. Essas noções são usadas na forma de terminologias específicas para se referir aos conceitos utilizados pela API e pela documentação. Por exemplo, em seu estudo, Thayer et al. (2021) utilizaram uma API chamada ThreeJS, que apoia a criação de animações 3D em browsers web, e solicitaram aos participantes de um experimento criarem uma animação de um anel reflexivo 3D girando acima de um oceano. Para eles completarem o que foi pedido, precisavam conhecer conceitos como eixos em 3D, eixos de rotação, o nome técnico da forma que eles querem criar e noções de cor e luz. Nenhum desses conceitos são específicos da API escolhida, são conceitos de Computação Gráfica.

Fatos de execução são conhecimentos declarativos no formato de regras simplificadas sobre o comportamento de execução de uma API, suficiente para prever, compreender e explicar sua execução. Os fatos de execução podem ser descritos como a modelagem dos conceitos de domínio. Seguindo com o exemplo da ThreeJS, para completar os passos de criar a forma pedida no experimento, o desenvolvedor precisava saber que a propriedade object.position.y muda a posição do objeto, object.rotation.y muda a rotação, TorusGeometry é a função que cria a forma e ainda conhecer o funcionamento dos parâmetros. Também era necessário saber que todas as mudanças serão visíveis no próximo quadro renderizado.

Padrões de uso de API são usualmente manifestados como padrões de código que mostram como parte da API pode ser utilizada. Porém, estes padrões não são apenas código,

incluem também comentários explicando o que cada parte do código faz, baseados nos conceitos de domínio e fatos de execução. Ainda no exemplo da API ThreeJS, para um desenvolvedor adicionar uma forma, ele deve primeiro criar um objeto “Geometry” e “Material”, com esses dois objetos criar um “Mesh” e então adicionar esse último objeto à cena. Um padrão de uso de API nesse caso iria conter a ordem que se deve criar objetos para se ter um “Mesh”, que ThreeJS pode ser usada para adicionar objetos à cenas e onde mudar o código para adicionar diferentes formas.

2.2.3. A documentação como barreira para o aprendizado de ferramentas

Meng et al (2020) afirmam que, infelizmente, documentações de ferramentas têm sido frequentemente descritas como pobres e incompletas, o que faz com que a documentação seja um dos principais obstáculos para um desenvolvedor aprender uma nova API.

De acordo com uma pesquisa realizada por Robillard (2009) junto a desenvolvedores da Microsoft com o intuito de entender quais eram as maiores dificuldades ao aprender uma nova ferramenta, os principais obstáculos encontrados foram causados por recursos inadequados ou ausentes para aprender aquela API. O autor divide as respostas relacionadas a recursos em seis categorias: (i) problemas relatados sobre falta de exemplos adequados, (ii) problemas não especificados com a documentação (descritos pelo autor como queixas gerais sobre a documentação oficial), (iii) conteúdo faltando ou apresentado inadequadamente, (iv) falta de especificação sobre como utilizar a API para completar determinada tarefa, (v) recursos não disponíveis da forma desejada e (vi) documentação faltante ou inadequada sobre aspectos de alto nível da API, como justificativas para as decisões de projeto. Isso mostra que, ao desenvolver uma API, não se deve atentar somente à implementação da mesma, mas também aos recursos disponíveis para que seja possível aprendê-la.

2.3. Trabalhos Relacionados

Estudos recentes sobre a qualidade das documentações do desenvolvedor têm sido realizados e serviram de base para esse trabalho.

O estudo de van der Meij e Carroll (1995) não é específico sobre documentação de software, mas os autores propõem princípios e heurísticas de design para elaborar documentações minimalistas. Uma abordagem minimalista significa não gastar tempo e recursos ensinando às pessoas nada além do mínimo necessário para realizar o que elas

desejam realizar (NICKERSON, 1991). O primeiro princípio é sobre escolher uma abordagem orientada à ação. Isto porque quando as pessoas estão aprendendo querem sentir que estão progredindo, e isso normalmente significa realizar algo com o artefato/ferramenta em mãos. O segundo diz que se deve apresentar a ferramenta no domínio da tarefa. Quando um indivíduo busca apoio de uma ferramenta, ele está focado no objetivo da tarefa que deseja realizar. Por isso, é importante garantir que as instruções mostrem como fazer as ações e atingir os objetivos esperados. O terceiro princípio é dar suporte a erros. As instruções devem ter suporte ao reconhecimento e recuperação de erros. Para isso, deve-se ter informações sobre os erros em uma proporção de cerca de uma vez a cada três ações, essas informações devem ser fáceis de serem entendidas e rápidas de serem percebidas. Por último, a documentação deve oferecer suporte tanto para indivíduos que estão interessados em realizar uma leitura sequencial quanto aqueles que buscam localizar uma informação em particular. A maioria dos usuários não leem os manuais inteiros, eles leem somente o tópico relacionado ao que eles querem fazer, por isso, deve-se ser o mais objetivo possível e deve-se tentar garantir que um capítulo não precise de informações de outro.

Smart (2002) analisa o livro “Produzindo Informações Técnicas de Qualidade” (IBM, 1983) e sua versão revisada “Desenvolvendo Informações Técnicas de Qualidade” (Hargis et al., 1997), que tratam da qualidade das documentações. Em seu artigo, Smart apresenta as principais contribuições do livro de 1983 e suas limitações, além de propor estudos para a expansão da teoria apresentada. Na edição revisada de 1997, foram definidas três categorias para classificar a qualidade das documentações: *Facilidade de Uso*, *Facilidade de Entendimento* e *Facilidade de Descoberta*, com três características cada uma. As características relacionadas à Facilidade *de Uso* são *Orientação à Tarefa, Precisão e Completude.* Em *Facilidade de Entendimento* estão *Clareza*, *Concretude e Estilo*. Em *Facilidade de Descoberta* estão *Organização*, *Recuperabilidade* e *Eficácia Visual*.

Maalej e Robillard (2013) argumentam que para discutir sobre a qualidade da documentação deve-se, primeiramente, saber qual o conhecimento que ela contém e como esse conhecimento é organizado. Para isso, eles organizaram uma pesquisa sobre os padrões de conhecimento em uma documentação, usando JDK 6 e .NET 4.0 como objetos de estudo. Para cada uma das tecnologias escolhidas foram analisados quais tipos de conhecimento ela apresenta. Ao final do estudo, foram categorizados doze tipos de conhecimento: *Funcionalidade e Comportamento, Conceitos, Diretrizes, Objetivo e Justificativa, Atributos de Qualidade e Aspectos Internos, Controle de Fluxo, Estrutura, Padrões, Exemplos de Código, Ambiente, Referências* e *Não Informação*. Esse foi o primeiro trabalho que temos

conhecimento a apresentar uma perspectiva abrangente sobre os padrões de conhecimento em documentações.

Robillard et al. (2017) propõem em sua pesquisa o que denominam de documentação do desenvolvedor sob demanda. Esse tipo de documentação seria gerada automaticamente em resposta a uma consulta do usuário. Para isso, seria usada uma combinação de técnicas de extração de conhecimento em uma coleção de artefatos que incluem código fonte, postagens em fóruns de discussão e metadados de sistemas de rastreamento de tarefas. Eles apresentam as dificuldades de desenvolver esse novo formato de documentação, mas argumentam que é um potencial avanço em relação às que temos atualmente, pois segundo os autores não são prioridade no desenvolvimento, já que a documentação do desenvolvedor exige um alto esforço e tem um baixo retorno do investimento no curto prazo.

Meng et al. (2020) apresentam diretrizes e heurísticas para a elaboração de melhorias nos modelos de documentação que temos atualmente. Essas diretrizes são baseadas em análises da literatura técnica sobre o que os usuários dessas documentações precisam, incluindo o trabalho de van der Meij e Carroll (1995) citado anteriormente, tendo algumas heurísticas compartilhadas entre eles. Também foi feito um estudo experimental comparando uma documentação elaborada seguindo as diretrizes e heurísticas apresentadas e outra sem essas melhorias. O estudo tinha dois grupos de participantes, cada um usando uma dessas documentações, tendo que realizar um conjunto de tarefas em uma API desconhecida por eles. O objetivo da API é facilitar a integração de lojas online com os serviços de fornecedores de remessas e é baseado no paradigma REST. Com isso, observou-se o impacto da documentação otimizada em duas perspectivas. A primeira relacionada ao processo de localizar, estudar e compreender a informação fornecida, e a segunda de planejar, preparar e executar a tarefa. Foi verificado que o tempo que os participantes levaram para procurar e estudar as informações nas documentações foi similar, porém, os que estavam usando a documentação melhorada realizaram estas ações com menos erros e gastaram menos tempo na fase de planejamento e nas chamadas da API.

Thayer et al. (2021) realizaram um estudo sobre os principais elementos que compõem uma documentação e como eles auxiliam no aprendizado de uma API. Eles definiram três classes de conhecimento: *Conceitos de Domínio*, *Fatos de Execução* e *Padrões de Uso de API*. Nesse estudo eles avaliaram, com um grupo de voluntários, como cada um desses tipos de conhecimento impactam na realização de determinadas ações ao usar uma ferramenta de desenvolvimento de software até então desconhecida pelos participantes. Para isso, eles desenvolveram uma interface que apresentava exemplo de código com anotações. Nela, era

possível selecionar parte do código e ver sua documentação, separada pelas suas classes de conhecimento. Dessa forma foi possível verificar o efeito de cada tipo de informação. Após analisar a forma que os participantes resolveram as tarefas propostas, medindo a quantidade de edições no código e o progresso, e o que eles acharam das documentações, eles encontraram evidências que indicam que a documentação, se não for bem construída, pode se tornar sobrecarregada e não ser fácil de ser utilizada quando necessária.

3. Metodologia

Este capítulo apresenta as escolhas metodológicas feitas neste trabalho com a seguinte organização: a subseção 3.1 define quais os objetivos do estudo e como o mesmo está organizado, a subseção 3.2 descreve como foi feita a seleção dos casos e os apresenta. A subseção 3.3 apresenta a forma que a coleta dos dados utilizados neste trabalho foi realizada. A subseção 3.4 descreve como foram feitas as análises dos dados coletados para cada uma das Questões de Pesquisa apresentadas anteriormente na subseção 3.1.

3.1. Objetivos e Organização do Estudo

O objetivo principal desta pesquisa é caracterizar as documentações utilizadas em ferramentas de desenvolvimento encontrados na indústria, quais os tipos de conhecimentos são mais frequentemente encontrados nelas e como eles estão estruturados. A partir desses objetivos foram definidas três perguntas que devem ser respondidas neste trabalho.

Questão de Pesquisa 1 (QP1): Quais os diferentes tipos de conhecimento que são capturados nas documentações?

Questão de Pesquisa 2 (QP2): Quais as estruturas de documentação do desenvolvedor para tecnologias de desenvolvimento de software são utilizadas?

Questão de Pesquisa 3 (QP3): Qual a percepção dos desenvolvedores sobre a qualidade das documentações?

Esta investigação está organizada como um estudo de caso múltiplo holístico. Diferentes tipos de estudo de caso são estudo de caso holístico e estudo de caso incorporado, onde o holístico estuda o caso como um todo e o incorporado estuda múltiplas unidades de análise dentro de um único caso. A classificação do estudo de mais de um caso como holístico múltiplo ou incorporado vai depender do contexto e objetivos da pesquisa (YIN, 2003 apud RUNESON e HÖST, 2008). A unidade de análise, que nesta investigação corresponde a um caso, será a documentação de uma ferramenta de desenvolvimento de software. As questões de pesquisa serão respondidas a partir da análise dos diferentes casos em conjunto, de forma comparativa, a fim de que as conclusões sobre elas possam ser obtidas. Não há hipóteses ou proposições pré-definidas, já que se trata de um estudo de natureza exploratória.

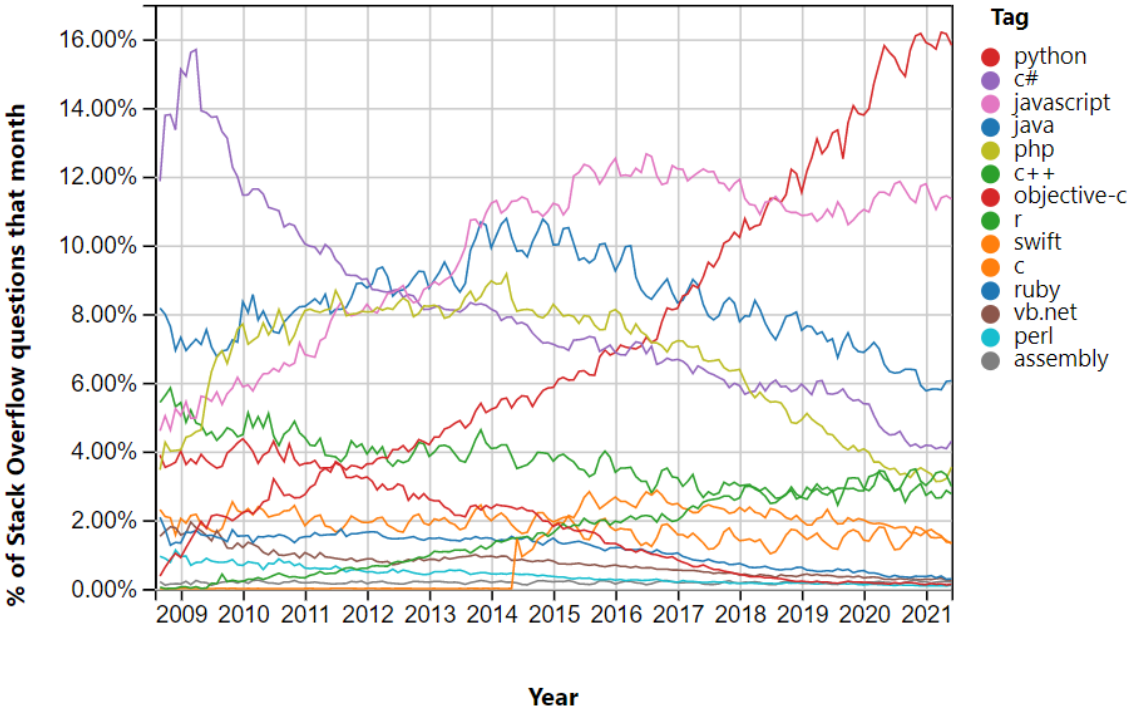
O enquadramento teórico para proposição, organização e análise deste estudo de caso são essencialmente as referências citadas no Capítulo 2 (VAN DER MEIJ e CARROLL, 1995, MENG et al., 2020, THAYER et al., 2021). O entendimento da documentação do desenvolvedor e das suas características são implicitamente influenciados por este enquadramento. Para os diferentes tipos de conhecimento presentes na documentação do desenvolvedor, o enquadramento teórico partirá dos tipos de conhecimento presentes na documentação proposta por Maalej e Robillard (2013) e nas três classes de conhecimento da teoria de Thayer et al (2021).

3.2. Seleção dos Casos

Os casos selecionados são documentações do desenvolvedor disponibilizadas no contexto de ferramentas de software disponibilizadas na indústria. A documentação deve estar disponível publicamente na Internet em sites relacionados às ferramentas. O critério de seleção foi direcionado a casos típicos, isto é, que não apresentam características distintas em relação ao que se observa usualmente na indústria.

Foram escolhidas ferramentas do escopo de sistemas Web, devido a predominância desses sistemas e pelo entendimento de que essas ferramentas são usadas também em outros tipos de sistemas (e.g., Mobile). Com o escopo definido, foi decidido que seriam utilizadas três linguagens de programação e, para cada uma delas, duas ferramentas para serem utilizadas. A plataforma de discussão de ferramentas de desenvolvimento *StackOverflow* foi consultada para definir quais ferramentas seriam utilizadas para a análise. O *StackOverflow* realiza pesquisas para saber quais as linguagens e ferramentas mais procuradas pelos seus usuários. Verificando a Figura 1, que representa os resultados da última pesquisa, foram selecionadas as três linguagens com mais perguntas no ano de 2021: Python, Javascript e Java.

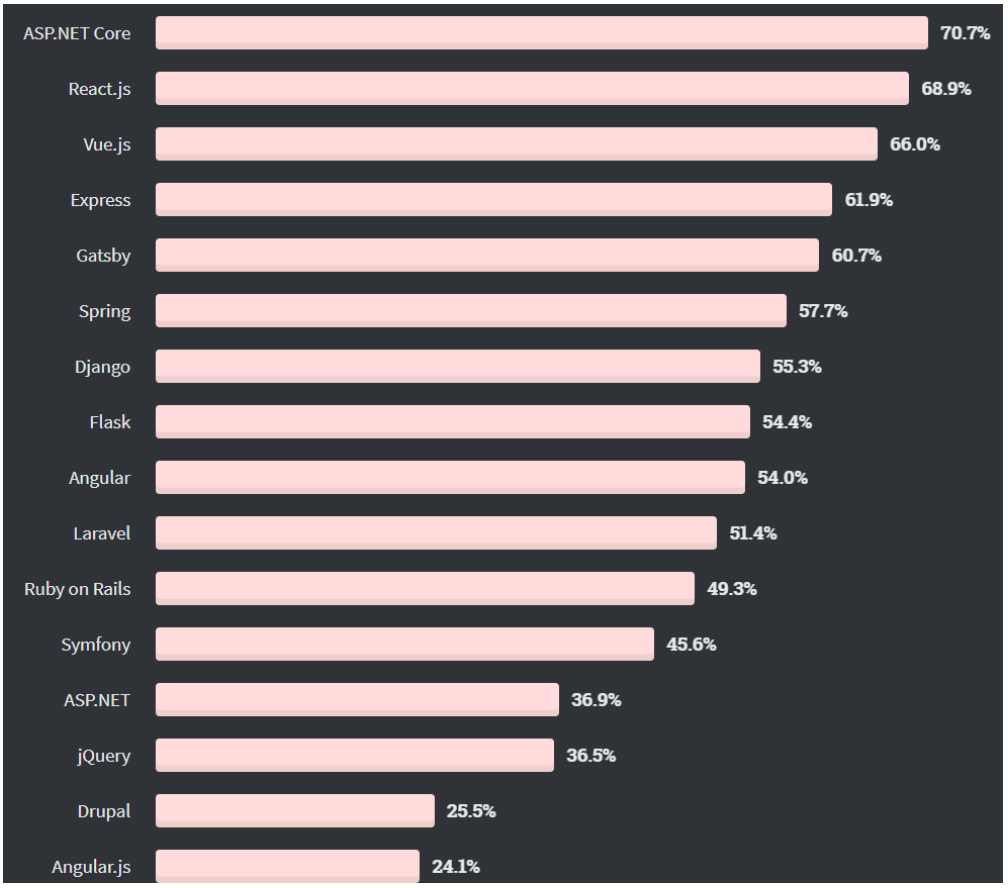
Figura 1: Evolução do uso de tecnologias ao longo do tempo com base no uso de suas tags desde 2008.



Fonte: Stack Overflow (2021).

Ainda seguindo as pesquisas realizadas pelo *StackOverflow,* também foi utilizado o levantamento observado na Figura 2 do uso de frameworks web. Por questões de organização do estudo, outros pontos foram levados em consideração na hora de escolher as ferramentas além da sua popularidade. Como cada linguagem deveria ter duas ferramentas, caso o tamanho da documentação da primeira ferramenta escolhida fosse muito extensa, a segunda deveria ser relativamente menor. Por isso, no caso da linguagem Java, como as ferramentas mais populares possuíam documentação extensa, foi selecionada a biblioteca SLF4J, uma biblioteca simples sobre a qual a autora já tinha conhecimento.

Figura 2: Percentual de desenvolvedores que estão desenvolvendo com a linguagem ou tecnologia e manifestaram interesse em continuar desenvolvendo com ela.



Fonte: Stack Overflow (2020).

Pensando nesses fatores, e levando em conta as linguagens mais usadas, foram escolhidos os seguintes casos: Para a linguagem Python: Django e Flask. Para Javascript: React.js e jQuery. Para o Java: Spring e SLF4J. Uma breve descrição destas ferramentas pode ser encontrada na Tabela 1.

Tabela 1 - Descrição das ferramentas analisadas neste trabalho

Nome	**Descrição**	**Onde Encontrar**
Spring	O Spring Framework fornece um modelo abrangente de programação e configuração para aplicativos empresariais modernos baseados em Java.	Spring Framework
SLF4J	Serve como uma abstração para vários frameworks de log, permitindo ao usuário conectar o framework desejado no momento do deploy.	SLF4J
React	Uma biblioteca do JavaScript para criar interfaces.	React – A JavaScript library for building user interfaces (reactjs.org)
jQuery	jQuery é uma biblioteca JavaScript que facilita atividades como a passagem e manipulação de documentos HTML, manipulação de eventos, animação e Ajax.	jQuery
Django	Django é um framework web Python de alto nível que incentiva o desenvolvimento rápido e um design limpo e pragmático.	The web framework for perfectionists with deadlines | Django (djangoproject.com)
Flask	Flask é um framework web, projetado para tornar os primeiros passos rápidos e fáceis, com a capacidade de escalar para aplicativos complexos.	Flask | The Pallets Projects

Fonte: Dados da Pesquisa.

3.3. Procedimentos e Coleta de Dados

Os dados foram coletados a partir das documentações disponibilizadas publicamente. Os procedimentos de coleta foram planejados de forma parcialmente estruturada. Para coleta e posterior análise, foram consideradas somente as documentações de alto nível. Dessa forma, as documentações de baixo nível que podem vir a existir em relação às ferramentas e tecnologias selecionadas não foram examinadas.

Para a QP1 a coleta foi feita de forma não estruturada, foram coletadas informações do conteúdo de cada seção das documentações. Para a QP2 foram observados os nomes das seções e subseções além do propósito de cada uma delas, essa coleta foi estruturada. Para

ambas as Questões de Pesquisa a coleta foi feita em julho de 2021 acessando os sites onde estavam disponibilizadas as documentações. Também de forma não estruturada e relacionada à QP3, foi feita uma coleta de dados adicionais na plataforma StackOverflow. Esta última foi realizada por meio da busca por perguntas feitas sobre as ferramentas analisadas que mencionaram a documentação. Para coletar esses dados foi feita uma pesquisa na plataforma usando os termos "[ferramenta] documentation" e foram analisados os primeiros 45 resultados, ordenados por mais relevantes. A coleta para essa Questão de Pesquisa foi feita em agosto de 2021.

Os dados foram armazenados para cada caso de forma separada e preparados para análise. A coleta estruturada foi armazenada na forma de um modelo de árvore que captura a hierarquia de seções utilizadas na documentação. As informações restantes foram armazenadas na forma de texto, em tabelas e com referências para onde foram retiradas de forma a manter a rastreabilidade.

3.4. Análise

Para a QP1, os dados foram analisados tomando como base a taxonomia proposta por Maalej e Robillard (2013) e as três categorias de conhecimento da teoria de Thayer et al (2021).

Tabela 2 - Descrição da taxonomia de Maalej e Robillard (2013)

Tipo de conhecimento	Descrição
Funcionalidade e Comportamento	Descreve o que a API faz em termos de funcionalidades e o que acontece quando ela é usada.
Conceitos	Explica o significado dos termos usados para descrever elementos da API e conceitos de design ou domínio usados ou implementados pela API.
Diretrizes	Especifica o que usuários podem ou não fazer com um elemento da API. São contratos explícitos de utilização da API.
Objetivo e Justificativa	Explica o propósito de fornecer um elemento ou a razão de uma determinada decisão de design.
Atributos de Qualidade e Aspectos Internos	Descreve os atributos de qualidade da API, também conhecidos como requisitos não-funcionais. Também se aplica a informações sobre a implementação interna da API que está apenas indiretamente relacionada ao seu comportamento observável.
Controle de Fluxo	Descreve como a API gerencia o fluxo de controle. Por exemplo, descrevendo quais eventos causam um determinado *callback* ser acionado.
Estrutura	Descreve a organização interna de um elemento (ex: classes ou métodos), informações sobre hierarquias de tipo ou como os elementos estão relacionados entre si.
Padrões	Descreve como obter resultados específicos com a API.
Exemplos de Código	Fornece exemplos de código de como usar e combinar elementos para implementar certas funcionalidades.
Ambiente	Descreve aspectos relacionados ao ambiente no qual a API é utilizada, mas não à API diretamente.
Referências	Inclui qualquer indicação para documentos externos, seja na forma de hiperlinks, marcados ”ver também”, ou menções de outros documentos.
Não Informação	Uma seção da documentação contendo qualquer frase completa ou fragmento autocontido de texto que fornece apenas texto não informativo.

Fonte: Maalej e Robillard (2013)

Tabela 3 - Descrição das classes de conhecimento

Classes de conhecimento	**Descrição**
Conceitos de domínio	Noções abstratas ou do mundo real que existem fora da API e que ela tenta modelar.
Fatos de Execução	Conhecimentos declarativos na forma de regras simplificadas sobre o comportamento de execução de uma API, suficiente para prever, compreender e explicar sua execução.
Padrões de Uso de API	São formas de padrão de código que mostram como parte da API pode ser utilizada.

Fonte: Thayer et al (2021)

A taxonomia de Maalej e Robillard (2013), descrita na Tabela 2, foi proposta para a documentação de baixo nível, que não estamos considerando neste trabalho. Por isso, para utilizá-la como ferramenta de análise verificamos sua adequabilidade ao tipo de documentação que estamos analisando, a documentação de alto nível. Para isso, foi feito um mapeamento em relação a taxonomia de Maalej e Robillard (2013) e as classes de conhecimento da teoria de Thayer et al (2021), descritas na Tabela 3, para descobrir se existe uma relação entre elas.

Um mapeamento subjetivo entre os termos da taxonomia e classes de conhecimento da teoria serviu de base para examinar esta adequabilidade. Verificou-se que quase todos os tipos de conhecimento da taxonomia estão relacionados com uma das classes de conhecimento da teoria de Thayer et al (2021). *Conceitos de Domínio*, que são conhecimentos a API modela, contemplam os seguintes itens da taxonomia: *Conceitos, Ambiente e Referências*. *Fatos de Execução*, que representam as regras de uso da API, contemplam *Funcionalidade e Comportamento, Diretrizes, Atributos de Qualidade e Aspectos Internos, Controle de Fluxo e Estrutura*. Por fim, *Padrões de Uso de API*, ou seja, como utilizar a API, está relacionado com *Objetivo e Justificativa, Padrões* e *Exemplos de Código.* Não foi identificada nenhuma relação do item *Não Informação* da taxonomia com as classes de conhecimento. No entanto, ao realizar as classificações das documentações, foi visto a necessidade de criar uma nova categoria, *FAQ*. Nela, estão as perguntas mais frequentes daquela ferramenta. Essa nova categoria se relaciona com a classe de conhecimento *Conceitos de Domínio.*

Por meio deste mapeamento, pôde-se identificar que os conceitos da taxonomia se relacionam com as classes de conhecimento presentes nas documentações de alto nível. Por conta disso, julgou-se conveniente utilizar a taxonomia de baixo nível de Maalej e Robillard

(2013) para analisar os tipos de conhecimento presentes na documentação de alto nível dos casos selecionados.

Sendo assim, utilizando a taxonomia de Maalej e Robillard (2013), a análise foi feita classificando as seções das documentações de acordo com o termo da taxonomia que mais se adequa ao seu conteúdo. Mesmo que a seção trate de mais de um tipo de conhecimento, ela foi classificada com base no propósito principal.

Como as documentações são muito extensas e considerando as restrições de tempo para analisar todas por completo, foi definido que essa classificação tomaria como base a organização hierárquica das seções de um documento. Sendo assim, a classificação de tipos de conhecimento foi feita para as seções do primeiro nível dos documentos e suas subseções foram consideradas como pertencentes ao mesmo tipo. Conforme será visto abaixo, na descrição da análise para QP2, utilizaremos representações em árvores para analisar as documentações. Por isso, utilizamos "nível" como o nível hierárquico em que as seções são apresentadas. O documento, ou seja, a raiz da árvore é nível 0. Primeiro nível são as seções pertencentes ao documento, segundo nível, as subseções do primeiro nível, e assim por diante.

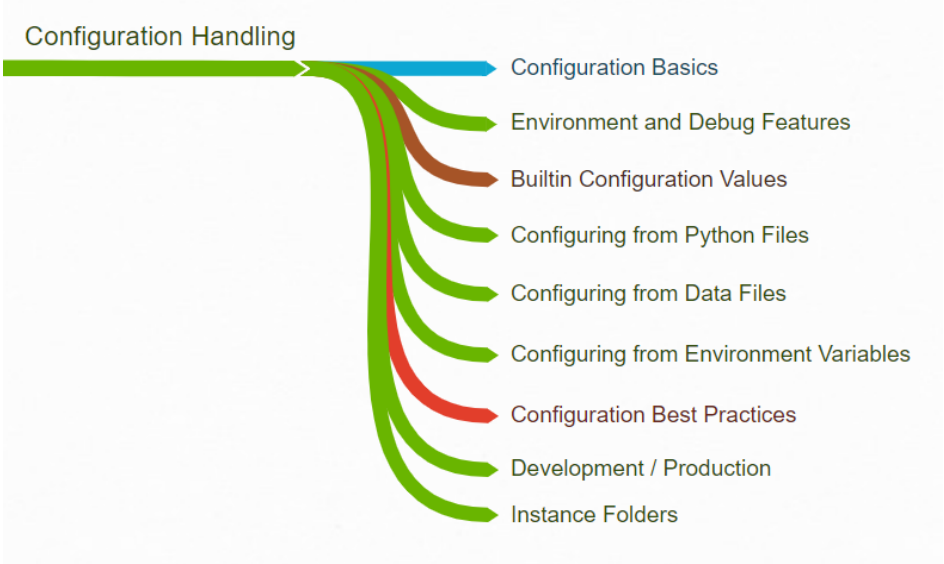
Dessa forma, a classificação realizada para a análise da QP1 se deu na perspectiva de primeiro nível da documentação - e não preciso ao ponto de identificar o tipo de conhecimento de cada subseção da documentação. Porém, para buscar atingir um maior nível de confiabilidade da análise, subseções de segundo nível também foram classificadas, mas apenas para uma amostra do total de seções do primeiro nível. Utilizando um nível de confiança de 90% e margem de erro de 20%, chegou-se a uma amostra de tamanho 15 para uma população total de 93 seções de primeiro nível que contivessem ao menos uma seção de segundo nível que pudesse ser classificada (35 seções de primeiro nível não tinham subseções). Para determinar a amostra foi utilizada a ferramenta Survey Monkey[7], que utiliza o z-score[8] no cálculo. Como a amostra deveria ter tamanho 15, para selecionar os casos que fariam parte dela foram sorteadas duas seções de cada uma das ferramentas analisadas. Somando, assim, 12 casos. Para os três casos restantes foram sorteadas três das ferramentas e, de cada uma delas, foi sorteada uma seção. Totalizando os 15 casos da amostra.

Um exemplo de como essa classificação foi feita pode ser observada na Figura 3 a seguir. A seção "Configuration Basics" foi classificada como *Funcionalidade e Comportamento,* "Builtin Configuration Values" como *Estrutura* e "Configuration Best Practices" como *Objetivo e Justificativa.* As seções restantes foram todas classificadas como

[7] https://www.surveymonkey.com/mp/sample-size-calculator/
[8] https://en.wikipedia.org/wiki/Standard_score

Ambiente. Com isso, pode ser observado que a maioria das subseções diz respeito a *Ambiente*, fazendo com que esse seja o propósito principal da seção de primeiro nível "Configuration Handling". Desta forma, classificamos a seção "Configuration Handling" como *Ambiente.*

Figura 3 - Exemplo de classificação de seção de segundo nível



Fonte: Dados da Pesquisa

O resultado da classificação desta amostra de seções de segundo nível, mostrou que a classificação anterior, feita no primeiro nível, tem nível de confiança aceitável. Das 15 seções analisadas, 12 haviam sido classificadas corretamente, o que equivale a 80%. Duas delas entraram no caso mencionado anteriormente neste capítulo de ter sido necessário criar uma categoria nova, porém na classificação de primeiro nível já havia sido percebida essa necessidade. As três restantes, que significam 20%, foram classificadas incorretamente e corrigidas.

Para a QP2, representações em árvore (ou floresta) das estruturas das documentações foram criadas para descrevê-las e compará-las. A descrição e comparação se deu em termos da ordem em que o conteúdo é disposto e a profundidade em que é apresentado. A análise de ambas dimensões foi facilitada pela estrutura em árvore.

A análise da QP3 também seguiu uma estratégia de classificação de conteúdo utilizando as dimensões de qualidade de Smart (2002), descritas na Tabela 4. Em cada pergunta no *StackOverflow* onde fosse identificada alguma relação com a documentação da ferramenta buscou-se caracterizar as dimensões de qualidade discutidas nelas.

Tabela 4 - Descrição das dimensões de qualidade de Smart (2002)

Fácil de Usar	
Orientação à Tarefa	Ajuda os usuários a completar tarefas relacionadas ao seu trabalho usando o produto.
Precisão	Não contém erros, as informações são verdadeiras.
Completude	Inclui todas as partes essenciais (mas apenas essas partes).
Fácil de Entender	
Clareza	Não contém ambiguidade ou complexidade desnecessária .
Concretude	Não contém abstrações; faltando exemplos apropriados, cenários e metáforas.
Estilo	Usa convenções corretas de escrita e escolha de palavras apropriadas.
Fácil de Encontrar	
Organização	Organiza o material de forma coerente de uma forma que faça sentido para o usuário.
Recuperabilidade	Apresenta informações de uma forma que permite aos usuários encontrar informações de forma rápida e fácil.
Eficácia Visual	Usa layout, ilustrações, cor, tipo, ícones e outros dispositivos gráficos para aprimorar o significado e a atratividade.

Fonte: SMART (2002)

4. Resultados e Discussão

A seguir serão apresentados os resultados deste estudo da seguinte forma: a seção 4.1 apresenta os resultados da Questão de Pesquisa 1, a seção 4.2 apresenta os resultados da Questão de Pesquisa 2 e a seção 4.3 apresenta os resultados da Questão de Pesquisa 3 A seção 4.4 apresenta uma discussão sobre os resultados, incluindo hipóteses para o que foi encontrado. Por último, a seção 4.5 descreve as ameaças à validade dos resultados obtidos.

4.1. Questão de Pesquisa 1

De forma a permitir obter uma visão geral dos tipos de conhecimento presentes nas documentações, a Tabela 5 traz os resultados quantitativos da análise das documentações. É importante pontuar que os dados presentes na tabela não são comparáveis entre as ferramentas, pois a quantidade total de seções de cada documentação é diferente. O objetivo principal dessa análise é a comparação do total de ocorrências de cada termo da taxonomia.

Tabela 5 - Relação entre as seções das documentações e a taxonomia utilizadas nesse trabalho

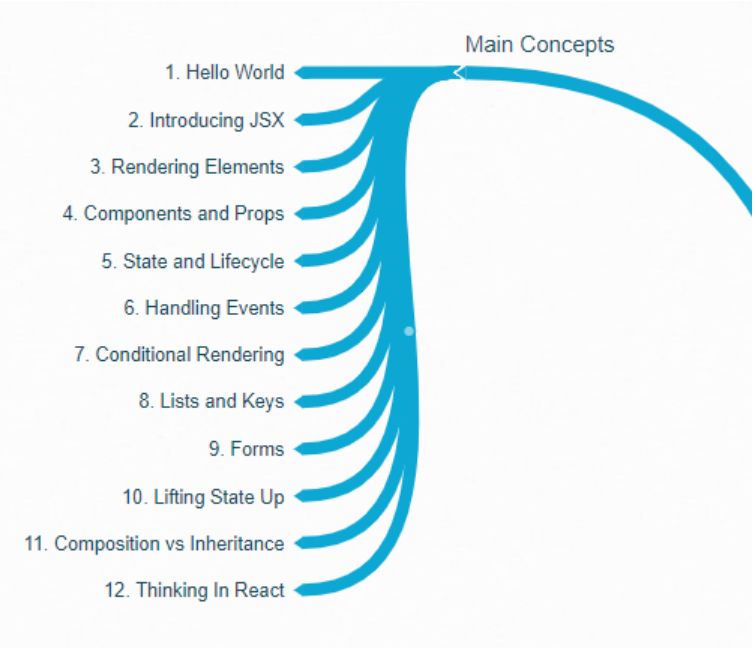
	Spring	SLF4J	React	JQuery	Django	Flask	Total
Funcionalidade e Comportamento	6	2	3	7	21	8	47 (39.1%)
Conceitos	0	0	0	0	0	0	0 (0%)
Diretrizes	0	1	0	0	0	0	1 (0.8%)
Objetivo e Justificativa	0	1	0	0	0	2	3 (2.5%)
Atributos de Qualidade e Aspectos Internos	2	0	1	1	1	4	9 (7.5%)
Controle de Fluxo	0	1	0	0	0	0	1 (0.8%)
Estrutura	0	0	0	1	0	0	1 (0.8%)
Padrões	0	1	1	1	20	7	30 (25%)
Exemplos de Código	2	0	3	0	3	1	9 (7.5%)
Ambiente	0	0	2	0	2	7	11 (9.1%)
Referências	1	0	1	0	1	1	4 (3.3%)
Não Informação	0	0	0	0	1	0	1 (0.8%)
FAQ	0	1	1	0	0	1	3 (2.5%)
Total:	11 (9.1%)	7 (5.8%)	12 (10%)	10 (8.3%)	49 (40.8%)	31 (25.8%)	120 (100%)

Fonte: Dados da Pesquisa

A partir dos dados da Tabela 5, é possível ver que o tipo de conhecimento mais comum nas documentações de alto nível é *Funcionalidade e Comportamento*. Isso porque a maioria das ferramentas possui seções em que são apresentadas as funcionalidades da API e

como utilizá-las. A Figura 4 apresenta um exemplo dessa seção. Nela é possível observar que os “Main Concepts” abrangem diferentes assuntos, porém, nenhum deles se aprofunda em conceitos e aspectos internos, o que mostra que essa seção tem o objetivo de mostrar o que pode ser feito com a ferramenta, e não as explicações por trás. Outro exemplo disso é o trecho retirado do subtópico “Introducing JSX” desta mesma seção: “Para renderizar um elemento React em um nó DOM raiz, passe ambos para ReactDOM.render(): [trecho de código para exemplificar].”.

Figura 4 - Trecho da árvore do React exemplificando o tipo *Funcionalidade e Comportamento*

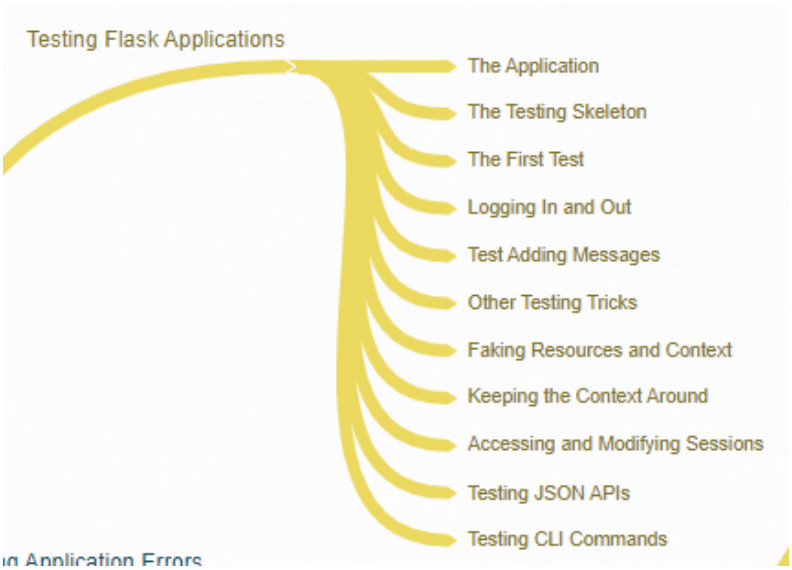


Fonte: Dados da Pesquisa

Padrões e *Exemplos de Código* também são relativamente comuns nas documentações selecionadas. Ambos tipos de conhecimento possuem semelhanças, pois são focados em instruir usando uma abordagem mais prática. Porém, *Padrões* são mais comuns e focam em demonstrar cenários e comportamentos específicos. Nas documentações este tipo usualmente manifesta-se em seções separadas para cada cenário, como pode ser observado na Figura 5 e no seguinte trecho retirado do subtópico “Test Adding Messages” do Flask: “Devemos também testar se a adição de mensagens funciona. Adicione uma nova função de teste como esta: [trecho de código onde a função mencionada foi adicionada] Aqui, verificamos se HTML é permitido no texto, mas não no título, que é o comportamento pretendido.”. Já os *Exemplos de Código* focam em cenários mais abrangentes, usando diversos recursos da ferramenta para chegar em um resultado mais complexo, como uma funcionalidade completa. Por isso, são normalmente observados nas documentações do tipo “Tutoriais”, que

apresentam um passo a passo de como construir e aprimorar um projeto, passando por diversos conceitos que são apresentados em outras seções como no exemplo da Figura 6. No trecho retirado de “Showing the Past Moves” fica evidente este processo de construção detalhado em etapas: “Uma vez que estamos registrando o histórico do jogo da velha, agora podemos exibi-lo para o jogador como uma lista de movimentos anteriores.[...] Em JavaScript, os arrays têm um método map() que é comumente usado para mapear dados para outros dados, por exemplo. [apresenta exemplo de código]”.

Figura 5 - Trecho exemplificando o tipo *Padrões*



Fonte: Dados da Pesquisa

Figura 6 - Trecho exemplificando o tipo *Exemplos de Código*

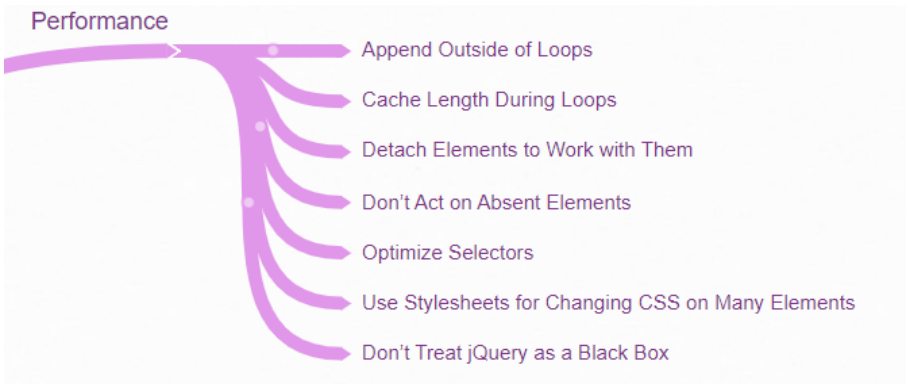
Fonte: Dados da Pesquisa

Dos tipos de conhecimento restantes, os que ainda têm alguma relevância em termos de frequência são *Ambiente* e *Atributos de Qualidade e Aspectos Internos*. *Ambiente* trata de aspectos do ambiente em que a ferramenta é utilizada, como o processo de instalação ilustrado na Figura 7. Na subseção “Install Python" o aspecto do ambiente fica mais evidente como observado no trecho “Django é um framework da Web Python. Veja ‘qual versão Python posso usar com Django?’ para detalhes. Obtenha a versão mais recente do Python em https://www.python.org/downloads/ ou com o gerenciador de pacotes do seu sistema operacional”. Neste trecho, vemos a preocupação em apresentar a linguagem em que a ferramenta é utilizada e como instalá-la. *Atributos de Qualidade e Aspectos Internos*, exemplificados na Figura 8, compreendem os requisitos não-funcionais da ferramenta e os aspectos que não estão diretamente relacionados com seu comportamento observável como, por exemplo, considerações de segurança e desempenho. Um exemplo é o seguinte trecho da subseção “Using Stylesheets for Changing CSS on Many Elements”: “Se você estiver alterando o CSS de mais de 20 elementos usando .css(), considere adicionar uma tag de estilo à página para um aumento de quase 60% na velocidade.”

Figura 7 - Trecho exemplificando o tipo *Ambiente*

Fonte: Dados da Pesquisa

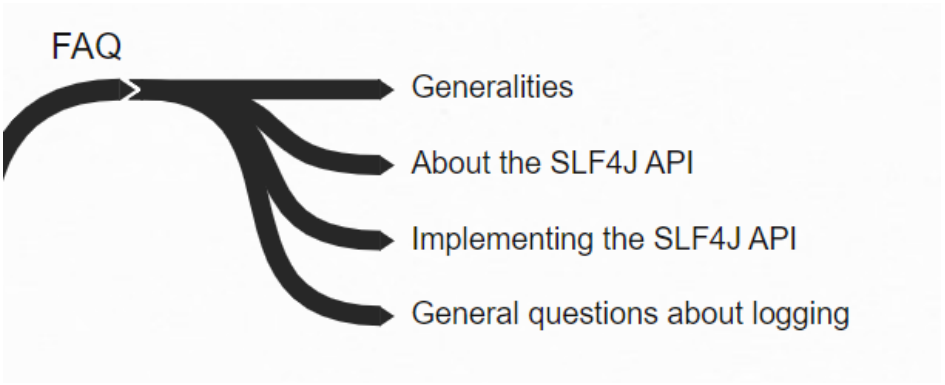
Figura 8 - Trecho exemplificando o tipo *Atributos de Qualidade e Aspectos Internos*



Fonte: Dados da Pesquisa

FAQ, categoria criada no contexto deste estudo, representa as seções que têm como objetivo responder as perguntas mais frequentes (FAQs). Um exemplo desta categoria pode ser visto na Figura 9.

Figura 9 - Trecho exemplificando o tipo *FAQ*



Fonte: Dados da Pesquisa

4.2. Questão de Pesquisa 2

A Questão de Pesquisa 2 tem como objetivo entender como os conhecimentos discutidos na Questão de Pesquisa 1 estão estruturados nas documentações estudadas. Para isso, a análise dessa pergunta foi dividida em duas partes: análise qualitativa e análise quantitativa. A análise qualitativa busca entender como as seções e subseções se relacionam. Enquanto a análise quantitativa busca descobrir se existe algum padrão em relação a profundidade e tamanho das documentações. Ambas as formas de análise foram realizadas com o auxílio das árvores criadas para cada documentação, como observadas nas Figuras 10 a 15.

As cores para cada tipo de conhecimento foram definidas de acordo com a Tabela 6 a seguir:

Tabela 6 - Tipos de conhecimento presentes nas documentações e as cores correspondentes

Tipo de conhecimento	Cor
Funcionalidade e Comportamento	Azul
Diretrizes	Roxo
Objetivo e Justificativa	Vermelho
Atributos de Qualidade e Aspectos Internos	Rosa
Controle de Fluxo	Verde claro
Estrutura	Marrom
Padrões	Amarelo
Exemplos de Código	Laranja
Ambiente	Verde escuro
Referências	Cinza
FAQ	Preto
Informação	Azul marinho

Figura 10 - Árvore representando a documentação do JQuery

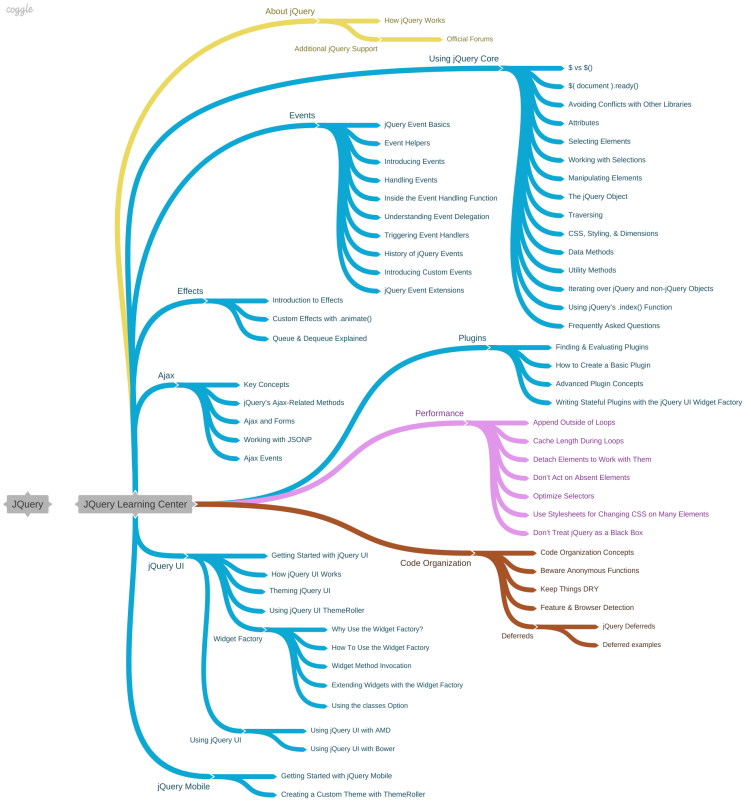


Figura 11 - Árvore representando a documentação do React

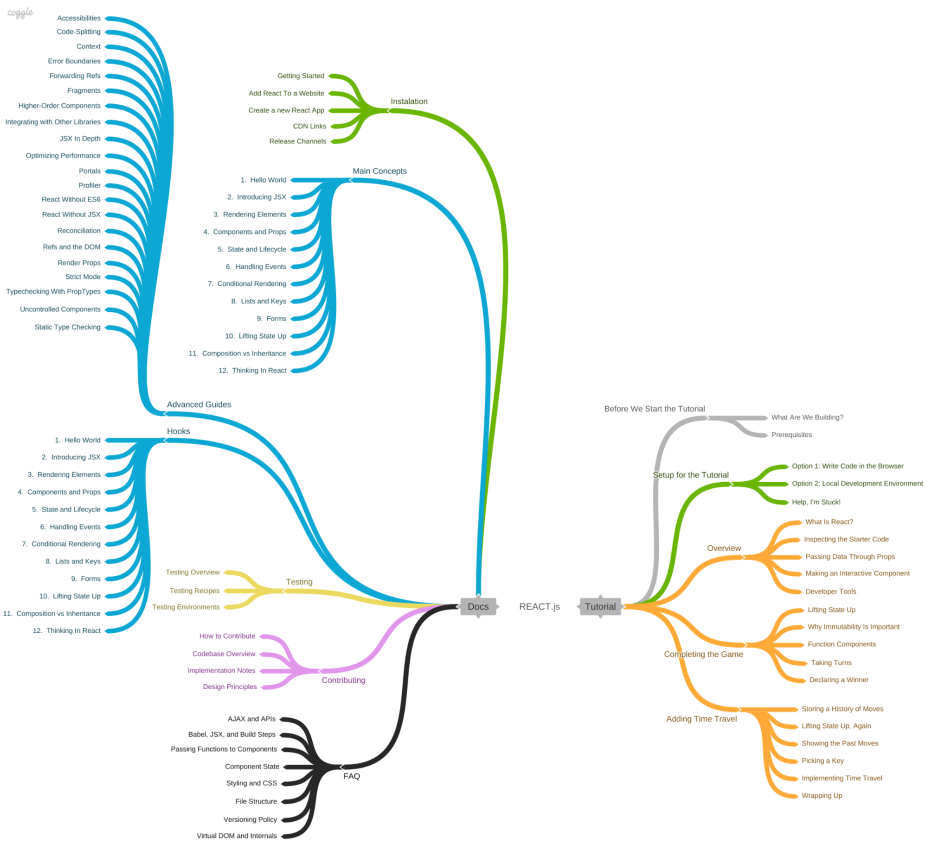


Figura 12 - Árvore representando a documentação do Django

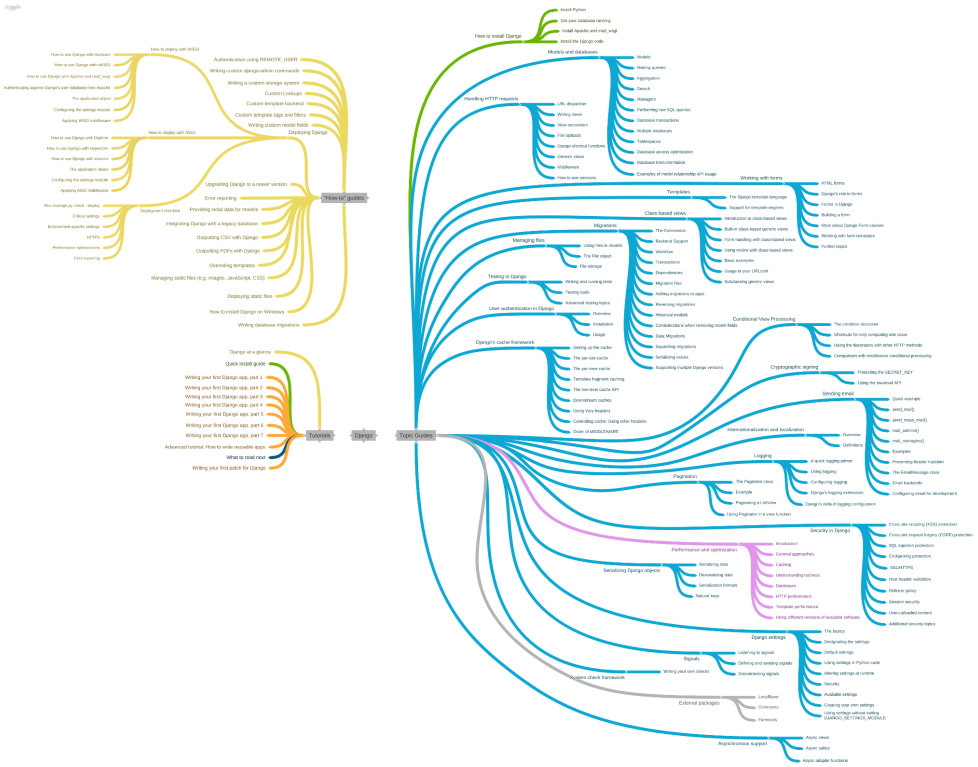


Figura 13 - Árvore representando a documentação do SLF4J

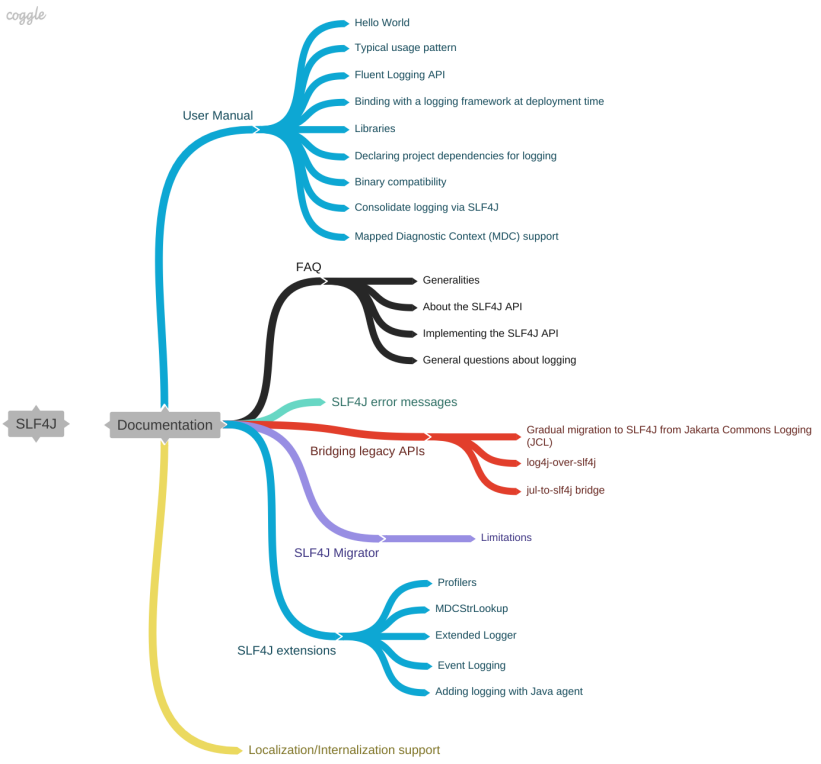


Figura 14 - Árvore representando a documentação do Spring

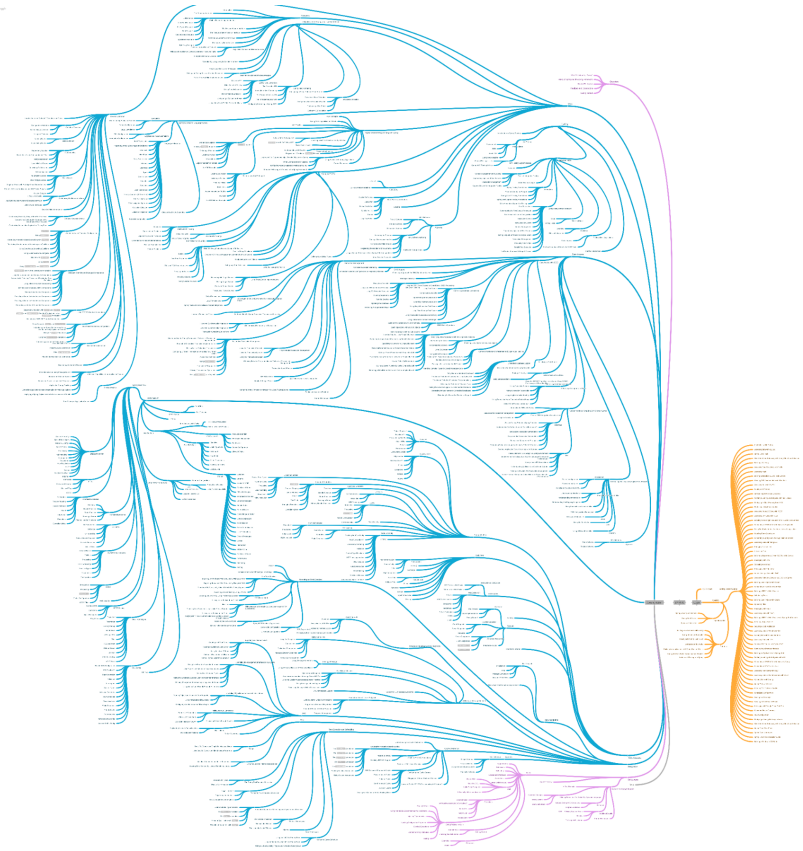
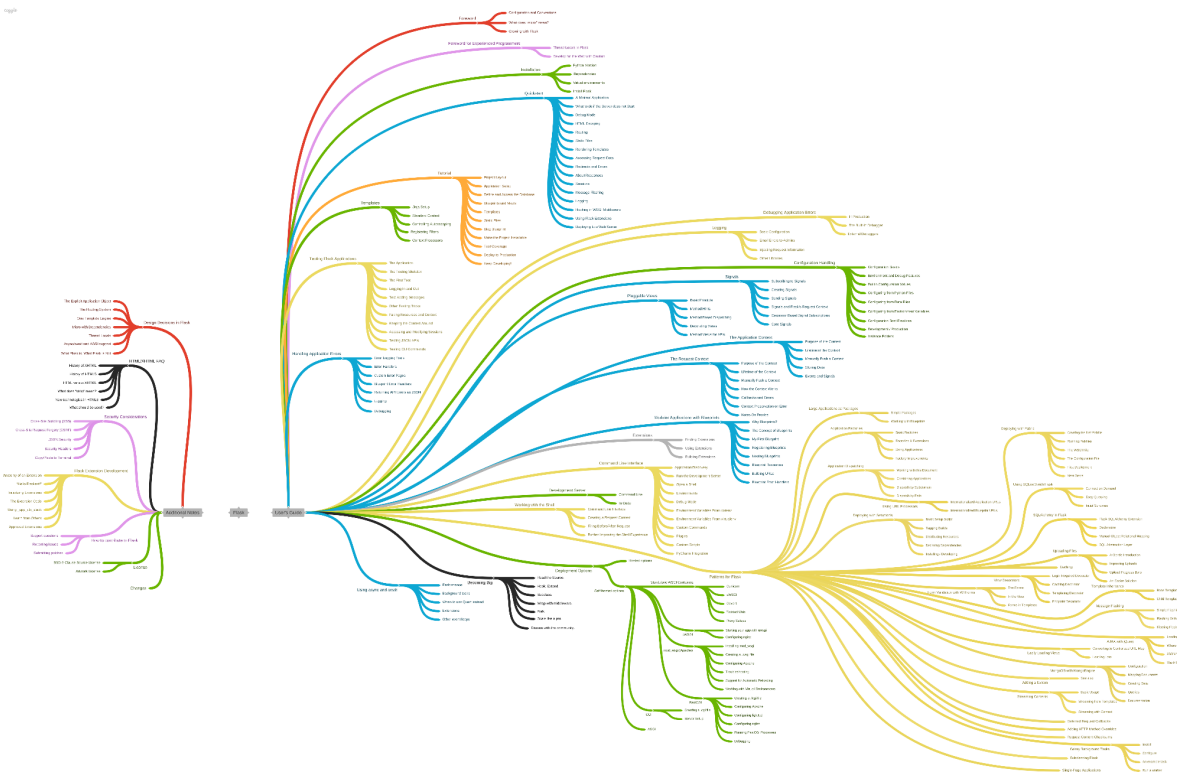


Figura 15 - Árvore representando a documentação do Flask



4.2.1. Análise Qualitativa

Essa análise está relacionada com os resultados encontrados na Questão de Pesquisa 1. Com base nos tipos de conhecimentos encontrados nas documentações analisadas, será explorado como eles estão organizados.

É possível observar que algumas das ferramentas analisadas apresentam mais de um documento. Esses documentos podem ter diferentes propósitos: apresentar os conceitos e funcionalidades da ferramenta, com explicações e aplicações; tutoriais em forma de passo a passo para o desenvolvimento de um pequeno projeto que utilize os diferentes elementos da ferramenta; entre outros. No caso do Flask, também existe um documento com informações adicionais, que em sua maior parte são direcionados ao desenvolvedor mais experiente. Já no caso do SLF4J, essas diferentes informações estão todas disponibilizadas em um mesmo documento. É importante ressaltar que os documentos em cada ferramenta podem estar presentes com diferentes nomenclaturas, mesmo que se tratem da mesma informação. Por exemplo, no React o documento que apresenta as funcionalidades se chama “Docs” enquanto que no Django se chama “Topic Guides”. Uma relação das ferramentas que apresentam mais de um documento pode ser observada na Tabela 6 a seguir.

Tabela 7 - Documentos das ferramentas que apresentam mais de um documento

Nome	Descrição
Spring - Learn	Apresenta tutoriais, ou seja, guias para completar determinadas ações
Spring - Documentation	Conceitos e funcionalidades existentes na ferramenta
React - Docs	Conceitos e funcionalidades existentes na ferramenta
React - Tutorial	Apresenta tutoriais, ou seja, guias para completar determinadas ações
Django - Tutorials	Apresenta tutoriais, ou seja, guias para completar determinadas ações
Django - Topic Guides	Conceitos e funcionalidades existentes na ferramenta
Django - How-Tos	Pequenos tutoriais para cenários mais específicos
Flask - Guide	Conceitos e funcionalidades existentes na ferramenta
Flask - Additional Notes	Informações adicionais sobre a ferramenta.

Analisando as árvores das documentações é possível perceber que algumas seções aparecem em mais de um dos casos selecionados. Todas as ferramentas apresentam, em pelo menos um de seus documentos, uma seção que explica como fazer a instalação e o setup da ferramenta, normalmente no início. Essa seção costuma ter ligação com o tipo de conhecimento *Ambiente* utilizado na classificação da QP1. Isso mostra que é importante apresentar para os usuários não só como usar a ferramenta, mas também como configurá-la.

Outro tópico que é apresentado em múltiplas documentações são os tutoriais. Três das seis ferramentas (Spring, React e Django) apresentam um documento focado nesse assunto. Nestas três ferramentas, a maioria das seções desse documento apresenta seções do tipo de conhecimento *Exemplos de Código*, com explicações em um formato passo a passo para no final completar alguma aplicação. O Spring apresenta múltiplos tutoriais de pequenos projetos separados em três níveis de dificuldade. Sendo que os mais avançados também apresentam explicações de arquitetura de software.

Conforme visto na QP1, o tipo mais comum de documentação de alto nível é a que apresenta os conceitos e funcionalidades básicas daquela ferramenta, com explicações de uso e pequenas aplicações em código-fonte para melhor entendimento. Ela está presente nas documentações de todas as ferramentas analisadas. Esse tipo costuma ser o foco principal de um documento, aparecendo no início e de forma mais extensa, tanto em relação a variedade de assuntos abordados como no detalhamento das informações.

Além dos conceitos básicos da ferramenta, alguns documentos também apresentam conceitos mais avançados. O React apresenta esses conceitos reunidos em um único tópico logo após os conceitos básicos, enquanto o SLF4J apresenta alguns tópicos separados sobre integrações[9] e extensões[10], com informações mais detalhadas de arquitetura. A documentação do Flask adota outra estratégia. Nela existe um documento chamado Additional Notes (Informações Adicionais) com informações que potencialmente interessam apenas aos desenvolvedores mais experientes. Neste documento, são apresentadas informações como as justificativas de implementação e arquitetura, preocupações de segurança e suas extensões.

Uma seção comum entre as ferramentas analisadas, presente em quatro delas (SLF4J, React, jQuery e Flask), é a FAQ (Frequently Asked Questions ou Perguntas Frequentes, em português). Nela são respondidas perguntas comuns dos usuários daquela ferramenta, desde dúvidas de conceitos, como de versionamento e de bibliotecas externas que possam vir a ser utilizadas. Foi observado que essa seção usualmente é localizada no final dos documentos, ou,

[9] SLF4J é uma ferramenta integrável com outras APIs, como o Log4j
[10] SLF4J é uma ferramenta extensível, contando com diferentes pacotes disponíveis

pelo menos, depois das informações principais. Ou seja, elas são adicionadas após as explicações dos conceitos da ferramenta ou após o passo a passo completo, dependendo do documento em que esta seção está inserida.

Informações sobre diferentes versões e releases foram encontradas em três das ferramentas. O Spring possui um documento inteiro para cada versão do framework. No jQuery são apresentados os métodos que foram depreciados, adicionados ou modificados em forma de documentação de baixo nível, que foi definido que não seria avaliado neste trabalho. Enquanto que no Flask as mudanças são apresentadas dentro do documento de Informações Adicionais.

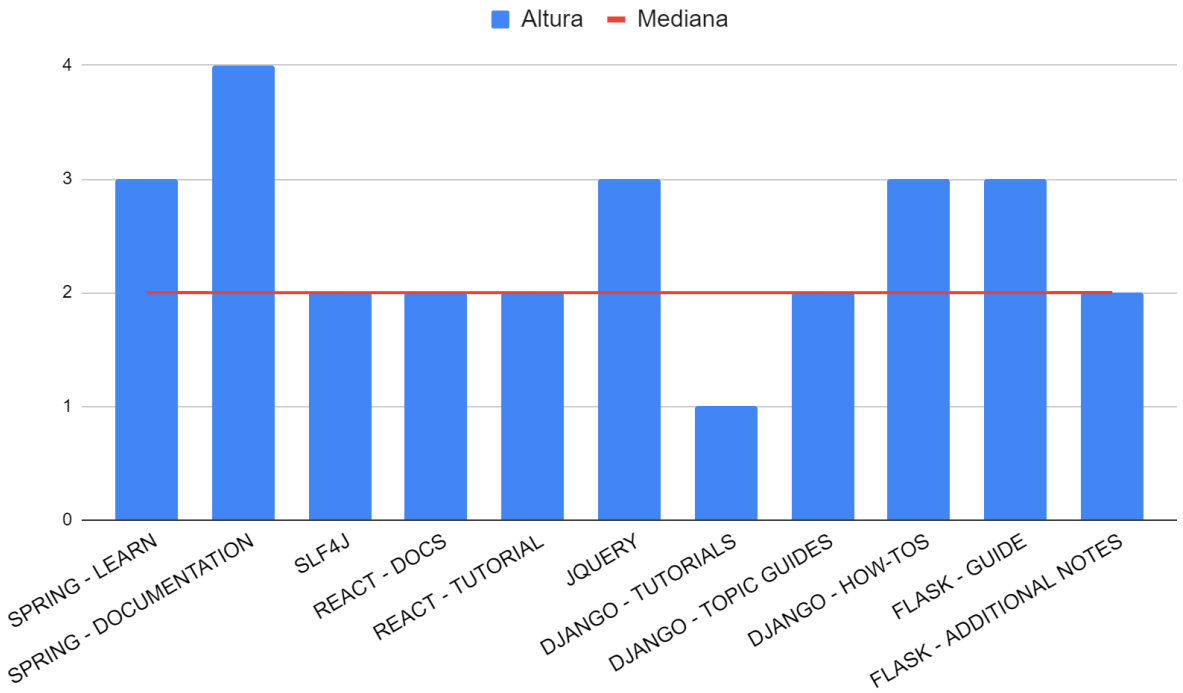
O React e o Flask foram os únicos em que foi possível encontrar uma seção para contribuir com a ferramenta, seja corrigindo algum problema ou melhorando a documentação. Em ambas, essa seção aparece entre os itens finais do documento em que está inserida.

4.2.2. Análise Quantitativa

Como mencionado na seção anterior, algumas ferramentas apresentam mais de um documento, como visto na Tabela 6. Para as análises nesta seção, serão considerados todos os documentos considerados de alto nível das ferramentas.

A primeira observação diz respeito à quantidade máxima de níveis que os documentos apresentam. Como pode ser observado na Figura 16, os documentos não possuem diferenças significativas nesse ponto. Parece existir um padrão para esse aspecto, com a maioria variando entre 2 e 3 níveis.

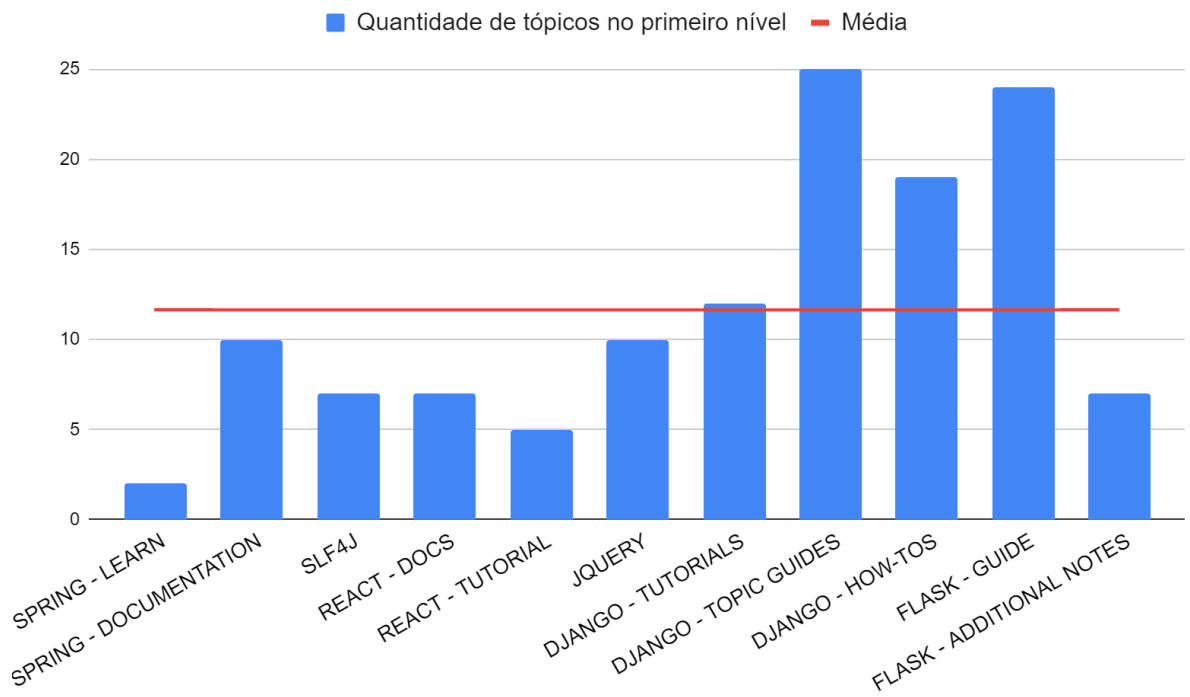
Figura 16 - Profundidade dos documentos



Fonte: Dados da Pesquisa

Outra análise que pode ser feita é em relação à quantidade de tópicos apresentados. Quanto mais tópicos um documento possui, potencialmente uma maior variedade de assuntos é tratada nele. Com essa análise é possível pensar em hipóteses sobre o que é necessário para uma documentação, se ela deve ser o mais abrangente possível ou se deve ser mais enxuta e priorizar apenas alguns tópicos mais importantes. Para isso, foi analisada a quantidade de seções de primeiro nível com base nas árvores representando os documentos. Os resultados estão descritos na Figura 17.

Figura 17 - Quantidade de tópicos de primeiro nível cada documento



Fonte: Dados da Pesquisa

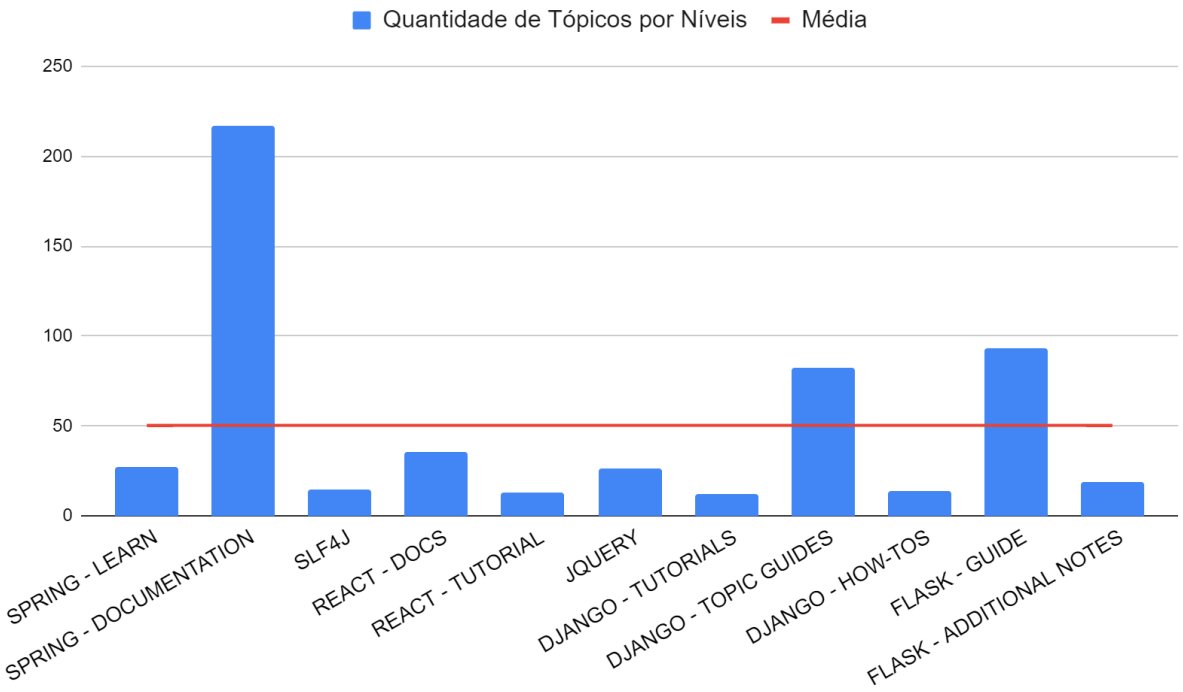
De acordo com a Figura 17, os documentos que focam em apresentar os conceitos e funcionalidades das ferramentas costumam ser os que possuem mais divisões de primeiro nível. Ou seja, eles apresentam mais tópicos do que os outros documentos, como os de tutoriais, por exemplo. Isto está relacionado ao fato, conforme visto na QP1, de que tipicamente a maior parte da documentação está associada aos conceitos e funcionalidades.

Ainda sobre a Figura 17, é interessante perceber que ferramentas do mesmo domínio podem apresentar diferenças grandes nesses resultados, indicando que o domínio da ferramenta não parece ser um fator para a organização do documento. Isso pode ser observado pelos casos do Spring, Django e Flask. Os três são frameworks web, porém o Django e o Flask possuem, no seu documento que apresenta conceitos e funcionalidades, mais de 20 tópicos de primeiro nível (o Django - Topic Guides apresenta 25 e o Flask - Guide apresenta 24). Enquanto que o mesmo documento do Spring, o Spring - Documentation apresenta 10 tópicos de primeiro nível.

Adicionando às análises anteriores, podemos também examinar a quantidade de seções em todo o documento considerando todos os níveis. Para isso, foi calculada a média de seções por nível, utilizando a quantidade total de seções dentre todos os níveis e a quantidade de níveis dos documentos. Observando a Figura 18, pode-se notar que os documentos que

apresentam os conceitos e funcionalidades são os que possuem mais tópicos por nível, ou seja, são potencialmente mais abrangentes que os das demais documentações.

Figura 18 - Quantidade média de tópicos por nível nos documentos



Fonte: Dados da Pesquisa

A diferença dessa análise para a anterior (quantidade de tópicos de primeiro nível) pode ser percebida nos exemplos do SLF4J e React-Docs. Segundo as Figuras 16 e 17, os dois documentos possuem a mesma profundidade e a mesma quantidade de tópicos de primeiro nível, porém, eles possuem uma diferença importante na quantidade média de tópicos por nível, o SLF4F apresenta uma média de 14,5 tópicos por nível e o React-Docs 36 tópicos por nível. Isso pode ser explicado observando as árvores apresentadas no início. Por elas, pode-se perceber que o documento React-Docs apresenta mais subníveis, ou seja, os tópicos estão mais fragmentados.

4.3. Questão de Pesquisa 3

A Questão de Pesquisa 3 foca em entender a percepção dos desenvolvedores sobre as documentações. Para obter uma resposta para essa pergunta, foram escolhidas duas ferramentas das seis analisadas neste estudo para realizar a coleta das opiniões. Com o auxílio da Tabela 5, elaborada para a Questão de Pesquisa 1, foram escolhidas as documentações que possuem a maior e a menor variedade de tipos de conhecimento. Três documentações

(Django, Flask e SLF4J) possuíam a maior variedade de tipos de conhecimento, com sete tipos cada uma. Neste caso, optou-se por analisar a mais popular (Figura 2). Com isso, as ferramentas escolhidas foram Django, a mais popular entre as com a maior variedade de conhecimentos, e JQuery, com quatro tipos de conhecimento, sendo assim a com menor variedade.

Com as ferramentas escolhidas, foi feita a coleta de dados para saber qual a percepção dos desenvolvedores sobre a documentação das ferramentas que utilizam. Foram verificados os 45 primeiros resultados de buscas feitas no StackOverflow para classificar os comentários de acordo com SMART(2002). Nas pesquisas feitas para ambas as ferramentas, a maioria das perguntas do StackOverflow que possuíam o termo “documentation” se deu apenas em razão de uma referência à documentação da respectiva tecnologia da resposta. Desta forma, não tinham comentários sobre sua qualidade. Outra classe comum de perguntas foram dúvidas sobre conceitos que não estão diretamente associados à ferramenta. Nestes casos, as respostas indicam que o conhecimento que falta não é da ferramenta em questão, mas da linguagem usada ou de alguma outra ferramenta utilizada.

A partir da identificação das situações mencionadas acima, foram selecionadas as perguntas e respostas no StackOverflow que concretamente tinham relação com algum aspecto de qualidade da documentação da tecnologia, descritos na Tabela 4 do Capítulo 3.
Para o JQuery restaram nove perguntas e no caso do Django sobraram cinco que foram classificadas de acordo com o aspecto de qualidade a qual as perguntas se relacionam, como mostra a Tabela 7 a seguir.

Tabela 8 - Quantidade de ocorrências dos aspectos ausentes nas documentações

	JQuery	Django
Fácil de Usar		
Orientação à Tarefa	0 (0%)	0 (0%)
Precisão	0 (0%)	0 (0%)
Completude	3 (33.3%)	0 (0%)
Fácil de Entender		
Clareza	4 (44.4%)	2 (40%)
Concretude	1 (11.1%)	2 (40%)
Estilo	0 (0%)	0 (0%)
Fácil de Encontrar		
Organização	0 (0%)	1 (20%)
Recuperabilidade	1 (11.1%)	0 (0%)
Eficácia Visual	0 (0%)	0 (0%)

Fonte: Dados da Pesquisa

Como pode ser observado, em ambas as ferramentas a maior parte das perguntas teve relação com *Entendimento* do conteúdo apresentado, no JQuery representando cinco das nove ocorrências de problemas, sendo uma de *Concretude* e quatro de *Clareza*, e no Django quatro das cinco, sendo duas de *Concretude* e duas de *Clareza*. O Django, ferramenta que contém conteúdo mais abrangente de acordo com a análise da Figura 18, não apresentou perguntas em relação à *Facilidade de Uso*, diferentemente do JQuery, que apresentou três perguntas sobre a *Completude* da documentação. Ambas também apresentaram problemas de *Facilidade de Descoberta* em menor escala, com uma ocorrência para ambas ferramentas. No JQuery o problema foi percebido como associado à *Recuperabilidade* da informação desejada enquanto que no Django sobre a *Organização* da mesma.

4.4. Discussão

Após a análise dos resultados apresentados nas seções anteriores e os comparando com os estudos descritos no Capítulo 2, foram encontradas algumas implicações e aspectos importantes das documentações que serão discutidas nesta seção.

4.4.1. Implicações

Em relação a taxonomia de Maalej e Robillard (2013), foi observado que o termo com mais ocorrências nas documentações foi *Funcionalidade e Comportamento,* sendo o único conhecimento que apareceu em documentações de todas as ferramentas. É interessante observar que no estudo de Maalej e Robillard (2013), que avaliou documentações de baixo nível, esse também foi o termo mais presente. Em nosso entendimento, isto fortalece os resultados observados no trabalho anterior ao mesmo tempo em que generaliza o seu escopo.

Essa predominância do tipo de conhecimento *Funcionalidade e Comportamento* parece estar em desacordo com os princípios e heurísticas propostos por van der Meij e Carroll (1995). O primeiro princípio definido pelos autores diz respeito à dar preferência a uma abordagem que seja orientada à tarefa, ou seja, segundo eles, as documentações devem ser descritas em termos das ações possíveis com o produto de forma que os usuários possam agir imediatamente para desempenhar suas atividades. Porém, *Funcionalidade e Comportamento* possui o foco principal de apresentar as funcionalidades das ferramentas, sem necessariamente apresentar contextos práticos em que o desenvolvedor pode utilizá-las.

Tendo mapeado cada uma das classes de conhecimento de Thayer et al (2021) em relação à taxonomia de Maalej e Robillard (2013) no Capítulo 3, o resultado da Questão de Pesquisa 1 também pode ser usado para caracterizar as documentações em relação a essas classes. Foi observado que a classe mais comum nas documentações é a de *Fatos de Execução*, que compreendem os termos da taxonomia mais frequentes: *Funcionalidade e Comportamento, Diretrizes, Atributos de Qualidade e Aspectos Internos, Controle de Fluxo* e *Estrutura.* Por outro lado, *Conceitos de Domínio* não foi identificado nas documentações do SLF4J e jQuery. De acordo com Thayer et al (2021), quando uma classe de conhecimento está faltando, os desenvolvedores gostariam de tê-la. Isso significa que, segundo a teoria dos autores, estas duas ferramentas apresentam deficiências de aprendizado em suas documentações.

4.4.2. Aspectos importantes da documentação

Depois de analisar todos os resultados das Questões de Pesquisa apresentadas no início deste Capítulo, foram encontrados alguns aspectos que foram considerados importantes de destacar. Em cima deles, é possível pensar em hipóteses que poderão ser estudadas no futuro.

Algumas das documentações das ferramentas selecionadas apresentam mais de um documento com diferentes propósitos. Essa separação da documentação em dois ou mais documentos parece ser comum, com quatro das seis ferramentas analisadas apresentando essa forma de organização. Separar os documentos de acordo com o propósito parece apresentar a vantagem de segmentar a documentação, tornando mais fácil para um desenvolvedor que sabe o que está procurando achar o que deseja. O que significa que essa separação de documentos potencialmente melhora a *Facilidade de Descoberta,* descrita em Smart (2002), da documentação. Além de que, assim, é possível saber de que forma o conteúdo do documento é apresentado. Porém, essa organização também apresenta desvantagens. Uma delas sendo que, quando o desenvolvedor não sabe exatamente o que procura e acessa a documentação em busca de auxílio na ferramenta, ele pode se sentir perdido. Porque não necessariamente ele tem preferência por alguma forma de apresentação do conteúdo, o que importa é conseguir assimilar o mesmo. Outro fator importante para argumentar contra essa separação é que a navegação entre os documentos pode se tornar trabalhosa e cansativa, principalmente em uma situação em que o desenvolvedor tenha interesse por tutoriais mas também queira entender melhor as funcionalidades e de que outras formas um elemento pode ser utilizado.

De acordo com o que foi discutido na seção anterior, a classe de conhecimento mais comum nas documentações é *Fatos de Execução.* Isso parece mostrar que o foco das documentações é em explicar o comportamento de execução das ferramentas, não em explicar os conceitos relacionados a ela. *Padrões de Uso de API* também apresenta uma presença significativa, o que dá a entender que mostrar como usar as ferramentas também é uma grande preocupação. Isso pode ser consequência da forma com que os desenvolvedores procuram informações. Muitas vezes com um foco maior em como usar, não necessariamente entender o comportamento por trás.

Sobre a quantidade de tópicos nas documentações analisadas por meio da Questão de Pesquisa 2, foi possível perceber que os documentos que apresentam os conhecimentos e funcionalidades costumam ter uma maior quantidade de tópicos por nível. Com esse resultado, e com o da Questão de Pesquisa 1, onde se identificou que a maior parte da

documentação parece estar relacionada a *Funcionalidade e Comportamento*, podemos supor que quanto mais tópicos por nível em um documento, mais abrangente ele está. Em adição a isso, podemos supor também que quanto mais tópicos por nível, mais detalhado está o conteúdo, já que ele fica mais dividido e, com isso, mais especificado.

Ainda em relação à estrutura das documentações, analisada na Questão de Pesquisa 2, foi possível perceber um padrão nas profundidades dos documentos, com a maioria variando entre dois e três níveis. Combinado com os resultados da Questão de Pesquisa 3, pode-se conjecturar se outros tipos de organização, com mais níveis, melhoraria a *Facilidade de Descoberta* das documentações. Com mais divisões de níveis, as seções teriam um escopo menor e mais específico, fazendo com que o conteúdo fique mais bem localizado. Porém, são necessárias maiores pesquisas sobre esse tema, analisando uma maior variedade de documentações e coletando a opinião dos desenvolvedores, para se afirmar o efeito da quantidade de níveis na *Facilidade de Descoberta* das documentações.

Outro ponto importante pode ser percebido ao observar a forma que os desenvolvedores procuram informações e comentam sobre as documentações. Na Questão de Pesquisa 3, foram analisadas 90 perguntas, 45 de cada uma das duas ferramentas selecionadas, para entender o que os desenvolvedores pensam sobre as documentações. De acordo com o resultado encontrado, pode-se pensar na hipótese de que uma maior variedade de tipos de conhecimento pode melhorar a qualidade de *Uso* da documentação. Porém, apenas essa variedade não impede outros problemas, como os de *Entendimento* e *Descoberta.* Para se ter uma maior clareza sobre essa questão, é necessário um estudo maior, analisando mais respostas e demais ferramentas.

4.5. Ameaças à Validade

Uma questão importante a ser pontuada em relação a esse estudo é a sua validade. Algumas decisões tomadas no desenvolvimento do mesmo podem ter influenciado nos resultados encontrados. Nesta seção serão discutidas essas decisões.

A primeira decisão tomada neste estudo foi como fazer a análise das seções das documentações. Foi definido que seriam analisadas somente as seções de primeiro nível. Isso apresenta alguns riscos já que, como visto, a maioria das documentações possuem entre 2 e 3 níveis e alguns não seriam analisados. Porém, foi decidido que os riscos não seriam muito significativos para um primeiro estudo. E, dessa forma, teria o benefício de ser possível

analisar a documentação com uma visão geral de sua estrutura, considerando a documentação como um todo.

Para tentar mitigar o risco de analisarmos somente o primeiro nível, foi feita também uma análise mais detalhada, observando o segundo nível, de algumas das seções de primeiro nível em uma amostra de 90% de nível de confiança e 20% de margem de erro, totalizando uma amostra de 15 casos. Essa classificação, porém, também apresenta um risco, já que, como foi mencionado no Capítulo 3, algumas seções apresentam mais de um tipo de conhecimento. A decisão tomada nesses casos foi descobrir qual o propósito principal daquela seção. Além disso, existe uma subjetividade nessas classificações que deve ser considerada.

A análise das perguntas do StackOverflow foi feita levando em consideração somente os primeiros 45 resultados. Sendo que em ambas as buscas foram encontrados pelo menos 500 resultados no total. Isso implica que o que foi encontrado nesta análise permite apenas apontar para alguns possíveis indícios.

Por último, é necessário considerar se os resultados deste trabalho podem ser generalizados para outros casos, ou seja, para as documentações de outras ferramentas. Como mencionado no Capítulo 3, os casos foram selecionados utilizando um critério direcionado para casos típicos, isto é, documentações que não apresentam características distintas dos observados normalmente na indústria. Desta forma, entende-se que os resultados aqui observados são potencialmente generalizáveis.

5. Conclusão

A proposta deste trabalho foi caracterizar as documentações do desenvolvedor de alto nível, com o objetivo de observar as informações que trazem, sua organização e suas deficiências.

5.1. Considerações finais

Esse trabalho realizou a caracterização das documentações baseadas em uma seleção de seis ferramentas de desenvolvimento de software. Foram criadas árvores para facilitar a visualização e classificação dos elementos dos documentos e, assim, responder às perguntas apresentadas no início deste trabalho.

A Questão de Pesquisa 1 tinha como objetivo caracterizar quais os diferentes tipos de conhecimento são capturados nas documentações. A classificação das seções dos documentos mostrou que o tipo de conhecimento mais presente foi *Funcionalidade e Comportamento*. Outro ponto importante foi a necessidade de criar uma nova classificação para tipos de conhecimento que não estavam sendo contemplados na taxonomia. Essa nova classificação foi denominada *FAQ* e contempla as perguntas mais frequentes daquela ferramenta.

A Questão de Pesquisa 2 tinha o objetivo de entender como as documentações do desenvolvedor para tecnologias de desenvolvimento de software são estruturadas. Para isso, foram criados gráficos com métricas de profundidade e quantidades de tópicos apresentados. Primeiramente foi observado que os documentos possuem muitas seções em comum, com muitas delas sendo apresentadas na mesma ordem e de formas parecidas. Por exemplo, as ferramentas que possuem FAQ e formas de contribuir apresentam essas seções no final dos documentos. Também foi visto que os documentos aparentam ter um padrão em relação a sua profundidade, com a maioria variando entre 2 e 3 níveis. Em relação a quantidade de tópicos, foi observado uma variação maior entre os documentos analisados, com os documentos apresentando médias de tópicos por nível bem diferentes entre si. Mesmo que as quantidades de níveis e de tópicos de primeiro nível sejam as mesmas.

Além de observar os documentos, também foi verificado o que os desenvolvedores comentam sobre as documentações que foram classificadas. Essa análise tinha o objetivo de responder a Questão de Pesquisa 3, sobre a percepção dos desenvolvedores sobre a qualidade

das documentações. Após ser feita uma seleção de duas ferramentas (jQuery e Django), foi feita uma pesquisa na plataforma *StackOverflow* com o nome da ferramenta e o termo "documentation" para se ter os dados necessários. Com essa busca foi possível observar que as documentações parecem não contemplar todas as dimensões de qualidade de SMART(2002). Sendo que a que apresentou indícios de problemas foi a *Facilidade de Entendimento.*

5.2. Limitações e Trabalhos futuros

Ao realizar um estudo podem surgir limitações que irão influenciar nos resultados obtidos. No caso deste estudo, a principal limitação encontrada estava relacionada com o tempo. Por esta restrição, não foi possível realizar a classificação de todas as seções de todos os documentos das ferramentas selecionadas. Por causa disso, foi necessário tomar decisões que podem ter influenciado nos resultados, como discutido no Capítulo 4. Da mesma forma, esta restrição também influenciou na seleção das perguntas na plataforma *StackOverflow.* Sendo definido que seriam analisados somente os 45 primeiros resultados de cada busca.

Algumas sugestões para trabalhos futuros são realizar uma nova classificação dos tipos de conhecimento apresentados nas documentações, mas em todos os níveis. Dessa forma, seria possível aumentar a confiança nos resultados deste trabalho. Como existe uma grande quantidade de dados a serem analisados nesse cenário, considerando todos os tópicos e subtópicos das documentações, uma abordagem interessante seria utilizar algoritmos de mineração de texto para realizar a extração e classificação.

Outro ponto importante é que existe a possibilidade de que os diferentes termos da taxonomia de Maalej e Robillard (2013) e as diferentes classes de conhecimento de Thayer et al (2021) possuam relevâncias distintas dependendo do domínio ou tipo de aplicação da ferramenta. Por isso, é necessário um novo estudo para analisar essa hipótese e comparar com os resultados desse trabalho, que trata dos dados de forma bruta, ou seja, trata todos os tipos e classes de conhecimento como tendo a mesma importância.

Além disso, procurar outras formas de analisar o que os desenvolvedores percebem sobre a documentação, pois 45 perguntas no *StackOverflow* representam uma amostra reduzida da quantidade do universo de desenvolvedores. Uma sugestão é realizar um estudo mais abrangente para descobrir qual a opinião dos desenvolvedores sobre a organização das documentações. Dessa forma seria possível validar se, por exemplo, a presença de mais de um documento apresenta mais vantagens ou desvantagens.

Além disso, também há a expectativa de que os resultados aqui apresentados possam servir como base de um diagnóstico sobre a documentação atual das ferramentas. Isto permitirá buscar avaliar formatos alternativos de documentação que possam se mostrar eventualmente mais bem aceitos pelos desenvolvedores do que os utilizados atualmente.

Referências Bibliográficas

THAYER, Kyle; CHASINS, Sarah E.; KO, Amy J. A Theory of Robust API Knowledge. **ACM Transactions on Computing Education**, v. 21, n. 1, p. 8:1-8:32, 2021.

MENG, Michael; STEINHARDT, Stephanie M.; SCHUBERT, Andreas. Optimizing API Documentation: Some Guidelines and Effects. *In*: **Proceedings of the 38th ACM International Conference on Design of Communication**. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–11. (SIGDOC '20). Disponível em: <https://doi.org/10.1145/3380851.3416759>. Acesso em: 25 jun. 2021.

SILLITO, Jonathan; BEGEL, Andrew. App-directed learning: An exploratory study. *In*: **2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)**. San Francisco, CA, USA: Institute of Electrical and Electronics Engineers (IEEE), 2013, p. 81–84.

SOHAN, S.M.; ANSLOW, Craig; MAURER, Frank. A Case Study of Web API Evolution. *In*: **2015 IEEE World Congress on Services**. New York, NY, USA: Institute of Electrical and Electronics Engineers (IEEE), 2015, p. 245–252.

ROBILLARD, Martin P.; MARCUS, Andrian; TREUDE, Christoph; *et al*. On-demand Developer Documentation. *In*: **2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)**. Shanghai, China: Institute of Electrical and Electronics Engineers (IEEE), 2017, p. 479–483.

MOTROC, Gabriela. GitHub survey: Incomplete documentation is the biggest problem encountered in open source. Disponível em: <https://jaxenter.com/github-survey-open-source-134645.html>. Acesso em: 22 maio 2021.

PARNIN, Chris. Api Documentation. Disponível em: <http://blog.ninlabs.com/2013/03/api-documentation/>. Acesso em: 22 maio 2021.

ROBILLARD, Martin P. What Makes APIs Hard to Learn? Answers from Developers. **IEEE Software**, v. 26, n. 6, p. 27–34, 2009.

WULF, Jochen; BLOHM, Ivo. Fostering Value Creation with Digital Platforms: A Unified Theory of the Application Programming Interface Design. **Journal of Management Information Systems**, v. 37, p. 251–308, 2020.

VAN DER MEIJ, Hans; CARROLL, John M. Principles and Heuristics for Designing Minimalist Instruction. **Technical Communication**, v. 42, n. 2, p. 243–261, 1995.

BALTES, Sebastian; TREUDE, Christoph; ROBILLARD, Martin P. Contextual Documentation Referencing on Stack Overflow. **IEEE Transactions on Software Engineering**, p. 1–1, 2020.

DAGENAIS, Barthélémy; ROBILLARD, Martin P. Creating and evolving developer documentation: understanding the decisions of open source contributors. *In*: **Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering**. New York, NY, USA: Association for Computing Machinery, 2010, p. 127–136. (FSE '10). Disponível em: <https://doi.org/10.1145/1882291.1882312>. Acesso em: 1 jul. 2021.

NICKERSON, Raymond S. A Minimalist Approach to the "Paradox of Sense Making". **Educational Researcher**, v. 20, n. 9, p. 24–26, 1991.

GAGGERO, Giancarlo. Program Documentation. *In*: MUXWORTHY, D. T. (Org.). **Programming for Software Sharing**. Dordrecht: Springer Netherlands, 1983, p. 175–188. (Ispra Courses). Disponível em: <https://doi.org/10.1007/978-94-009-7145-5_12>. Acesso em: 3 jul. 2021.

Stack Overflow Developer Survey 2020. Stack Overflow. Disponível em: <https://insights.stackoverflow.com/survey/2020/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2020>. Acesso em: 4 jul. 2021.

Stack Overflow Trends. Disponível em: <https://insights.stackoverflow.com/trends?tags=java%2Cc%2Cc%2B%2B%2Cpython%2Cc%23%2Cvb.net%2Cjavascript%2Cassembly%2Cphp%2Cperl%2Cruby%2Cswift%2Cr%2Cobjective-c>. Acesso em: 4 jul. 2021.

ROBILLARD, Martin P.; DELINE, Robert. A field study of API learning obstacles. **Empirical Software Engineering**, v. 16, n. 6, p. 703–732, 2011.

RUNESON, Per; HÖST, Martin. Guidelines for conducting and reporting case study research in software engineering. **Empirical Software Engineering**, v. 14, n. 2, p. 131, 2008.

SMART, Karl L. Assessing quality documents. **ACM Journal of Computer Documentation**, v. 26, n. 3, p. 130–140, 2002.

MAALEJ, Walid; ROBILLARD, Martin P. Patterns of Knowledge in API Reference Documentation. **IEEE Transactions on Software Engineering**, v. 39, n. 9, p. 1264–1282, 2013.

FALBO, Ricardo; MENEZES, Crediné; ROCHA, A. Using Ontologies to Improve Knowledge Integration in Software Engineering Environments. In: **4th International Conference on Information Systems Analysis and Synthesis (ISAS).** Orlando (USA): ISAS '98, 1998.

Standard score. In: Wikipedia. [s.l.: s.n.], 2021. Disponível em: <https://en.wikipedia.org/w/index.php?title=Standard_score&oldid=1040522883>. Acesso em: 14 set. 2021.

ABRAHAMSON, Karl R., Importance of documentation, disponível em: <http://www.cs.ecu.edu/karl/3300/spr16/Notes/Documentation/documentation.html>. acesso em: 23 maio 2021

Apêndice 1 - Tabela com links para as documentações

SPRING - LEARN	Spring | Learn
SPRING - DOCUMENTATION	Spring Framework Documentation
SLF4J	http://www.slf4j.org/docs.html
REACT - DOCS	Getting Started – React (reactjs.org)
REACT - TUTORIAL	Tutorial: Intro to React – React (reactjs.org)
JQUERY	jQuery Learning Center
DJANGO - TUTORIALS	Getting started | Django documentation | Django (djangoproject.com)
DJANGO - TOPIC GUIDES	Using Django | Django documentation | Django (djangoproject.com)
DJANGO - HOW-TOS	“How-to” guides | Django documentation | Django (djangoproject.com)
FLASK	Welcome to Flask — Flask Documentation (2.0.x) (palletsprojects.com)