UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE CRATEÚS

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

FRANCISCA GINIELE DO NASCIMENTO PINHO

DESAFIOS E SOLUÇÕES NA DOCUMENTAÇÃO EM PROJETOS DE SOFTWARE LIVRE E DE CÓDIGO ABERTO: UM MAPEAMENTO SISTEMÁTICO DA LITERATURA

CRATEÚS

2024

FRANCISCA GINIELE DO NASCIMENTO PINHO

DESAFIOS E SOLUÇÕES NA DOCUMENTAÇÃO EM PROJETOS DE SOFTWARE LIVRE E DE CÓDIGO ABERTO: UM MAPEAMENTO SISTEMÁTICO DA LITERATURA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus de Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Orientador: Profa. Me. Simone de Oliveira Santos

Coorientador: Prof. Dr. Allysson Allex Araújo

CRATEÚS

2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

P723d Pinho, Francisca Giniele do Nascimento.
Desafios e soluções na documentação em projetos de software livre e de código aberto: um mapeamento sistemático da literatura / Francisca Giniele do Nascimento Pinho. – 2024.
50 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Crateús, Curso de Sistemas de Informação, Crateús, 2024.
Orientação: Profa. Ma. Simone de Oliveira Santos.
Coorientação: Prof. Dr. Allysson Allex Araújo.

1. Documentação de Software. 2. Software Livre e de Código Aberto. 3. Desafios. 4. Soluções. 5. Mapeamento Sistemático da Literatura. I. Título.

CDD 005

FRANCISCA GINIELE DO NASCIMENTO PINHO

DESAFIOS E SOLUÇÕES NA DOCUMENTAÇÃO EM PROJETOS DE SOFTWARE LIVRE E DE CÓDIGO ABERTO: UM MAPEAMENTO SISTEMÁTICO DA LITERATURA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus de Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Aprovada em:

BANCA EXAMINADORA

Profa. Me. Simone de Oliveira Santos (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Allysson Allex Araújo (Coorientador)
Universidade Federal do Cariri (UFCA)

Prof. Dr. Igor Scaliante Wiese (Coorientador)
Universidade Tecnológica Federal do Paraná (UTFPR)

Profa. Me. Lisieux Marie M. dos S. Andrade
Universidade Federal do Ceará (UFC)

Profa. Dra. Amanda Drielly Pires Venceslau
Universidade Federal do Ceará (UFC)

À minha mãe (in memoriam), que foi e sempre será minha maior inspiração e fortaleza.

AGRADECIMENTOS

Aos meus irmãos, especialmente ao George Pinho, pelo apoio constante e incentivo diário. Ao meu amor, Paulo Rodrigues, pela complacência, motivação e por tornar essa trajetória mais leve.

Aos meus professores orientadores que me acolheram como orientanda, compartilhando comigo seus vastos conhecimentos e entusiasmo como pesquisadores. Agradeço pelos valiosos feedbacks, pela paciência e pela excelência.

Por fim, expresso minha gratidão a todos os professores que contribuíram para a minha formação acadêmica.

“Deixe-me ir, preciso andar, vou por aí a procurar
rir pra não chorar.”

(Cartola)

RESUMO

A documentação de software se revela um processo fundamental para o desenvolvimento de software de qualidade, pois auxilia as partes interessadas a utilizar, compreender, manter e implementar um software de forma produtiva. No entanto, ao se averiguar o contexto de projetos de software livre e de código aberto (em inglês, *free and open-source software* ou FOSS), pode-se constatar particularidades únicas as quais demanda de singular atenção como, por exemplo, pelo fato de haver a natureza voluntária com entradas e saídas de colaboradores de forma recorrente, diversidade de experiências, etc. Diante disso, através de Mapeamento Sistemático da Literatura (MSL), este trabalho tem como objetivo mapear, com base nos últimos dez anos de pesquisa científica (publicadas entre 2013 e 2023), os desafios e soluções em relação à documentação de software em projetos FOSS. A partir de um conjunto inicial de 1271 artigos, identificou-se 12 estudos primários para os quais tornou-se possível extrair cinco desafios (colaboração, qualidade, incompletude, manutenibilidade e categorização) e três perspectivas gerais de soluções investigadas (Uso Estratégico de README, Adoção de Inteligência Artificial e Ferramentas e Abordagens de Apoio). Como contribuições acadêmicas, esta pesquisa oferece um MSL revelando um conjunto de desafios e soluções relacionadas à documentação de software, um tópico ainda pouco explorado no contexto de pesquisa em FOSS. Sob o ponto de vista da prática, este trabalho promove uma reflexão sobre o uso das documentações em projetos FOSS, reverberando desafios e soluções que podem contribuir para o aperfeiçoamento da qualidade das documentações em projetos FOSS.

Palavras-chave: Documentação de Software. Software Livre e de Código Aberto. Desafios. Soluções. Mapeamento Sistemático da Literatura.

ABSTRACT

Software documentation proves to be a fundamental process for the development of quality software, as it helps interested parties to use, understand, maintain and implement software productively. However, when investigating the context of free and open source software projects (in English, *free and open-source software* or FOSS), unique particularities can be seen which demand singular attention, such as For example, due to the fact that there is a voluntary nature with employees coming and going on a recurring basis, diversity of experiences, etc. Therefore, through Systematic Literature Mapping (MSL), this work aims to map, based on the last ten years of scientific research (published between 2013 and 2023), the challenges and solutions in relation to software documentation in FOSS projects . From an initial set of 1271 articles, 12 primary studies were identified for which it was possible to extract five challenges (collaboration, quality, incompleteness, maintainability and categorization) and three general perspectives of investigated solutions (Strategic Use of README, Adoption of Artificial Intelligence and Supporting Tools and Approaches). As academic contributions, this research offers an MSL revealing a set of challenges and solutions related to software documentation, a topic still little explored in the context of FOSS research. From a practical point of view, this work promotes reflection on the use of documentation in FOSS projects, reflecting challenges and solutions that can contribute to improving the quality of documentation in FOSS projects.

Keywords: Software Documentation. Free and Open Source Software. Challenges. Solutions. Systematic Mapping Study.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

A documentação de software desempenha um papel salutar no desenvolvimento de software de qualidade (PLÖSCH *et al.*, 2014). Tal importância reside no fato de que ela auxilia as partes interessadas a utilizar, compreender, manter e desenvolver um sistema de maneira mais produtiva (AGHAJANI *et al.*, 2020; GAROUSI *et al.*, 2013). De acordo com Parnas (2010) e Forward (2002), a documentação de software é um artefato destinado a comunicar informações sobre o sistema de software ao qual está relacionado, sendo uma descrição escrita que pode ser utilizada como evidência. Assim, uma documentação útil e de qualidade é um dos fatores-chave na produção de software de alta qualidade (BRIAND, 2003). Portanto, espera-se que a documentação forneça informações precisas e úteis sobre os sistemas, tanto em projetos com código fechado (*closed source*) quanto em projetos com código aberto (*open source*).

Os projetos de software livre e de código aberto (em inglês, *free and open-source software* ou FOSS) são soluções nas quais os usuários podem distribuir, acessar, adaptar, modificar e redistribuir livremente o código-fonte para uso próprio e da comunidade (KLUG *et al.*, 2021). Assim, a comunidade de um projeto FOSS consiste em desenvolvedores e usuários que compartilham o interesse comum de colaborar na criação de soluções de software (KAUR *et al.*, 2022; TRINKENREICH, 2021). A adoção de um modelo FOSS tem se mostrado uma oportunidade estratégica para organizações em diversos domínios, pois oferece benefícios como acesso a programadores talentosos em qualquer lugar do mundo, potencial para revisão por pares e a capacidade de distribuir, adaptar e modificar livremente o código-fonte (FRANCO-BEDOYA *et al.*, 2017). Tal perspectiva estratégica promove às organizações um potencial aumento da inovação e qualidade do software desenvolvido (WEST; BOGERS, 2014).

Apesar das vantagens do modelo de desenvolvimento FOSS, como a flexibilidade de ter excelentes programadores de qualquer lugar do mundo e o potencial substancial de revisão por pares, ele também enfrenta particularidades exclusivas (MICHLMAYR *et al.*, 2005; KAUR *et al.*, 2022). Por exemplo, tem-se a natureza voluntária da participação, que pode levar a entradas e saídas de colaboradores, a falta de gerenciamento centralizado para direcionar os recursos e atributos dos desenvolvedores, a diversidade de habilidades e experiência dos envolvidos e a necessidade de recrutar e reter novos colaboradores (DING *et al.*, 2014b; LAKULU *et al.*, 2010). Diante de tais questões, a documentação de software, enquanto processo, acaba por ser impactada e, consequentemente, demanda uma atenção especial tendo em vista que promove benefícios distintos para o desenvolvimento de software, como apoio à comunicação assíncrona

para nivelar o conhecimento entre os colaboradores, bem como facilitar a gestão do conhecimento (MICHLMAYR *et al.*, 2005).

No entanto, embora a documentação de software seja um tópico amplamente investigado pela comunidade de Engenharia de Software (ZHI *et al.*, 2015; SOUZA *et al.*, 2005; GAROUSI *et al.*, 2013), constata-se uma lacuna de pesquisa em relação ao mapeamento sistemático das desafios e soluções de documentação de software em projetos FOSS. Diante dessa oportunidade, e com o objetivo de compreender o estado-da-arte sobre essa interseção de estudos envolvendo o desenvolvimento *open source* e a documentação de software, este trabalho propõe investigar a seguinte questão de pesquisa: *Nos últimos dez anos, quais são os desafios e soluções em relação à documentação de software em projetos FOSS?* Em especial, a opção por uma janela temporal de dez anos se justifica a partir da decisão em focar nas perspectivas recentes que enquadram o cenário de engenharia de software de projetos FOSS.

Considerando o escopo da questão de pesquisa exposta anteriormente, este trabalho adota uma abordagem metodológica baseada em um Mapeamento Sistemático da Literatura (MSL). Em especial, o MSL se mostra oportuna, uma vez que seu objetivo é revisar, classificar e estruturar artigos relacionados a um campo de pesquisa específico (PETERSEN *et al.*, 2008). O uso do MSL se justifica pelo seu foco em delinear um campo específico de pesquisa, fornecendo uma visão geral sobre o tema e possibilitando a categorização do tópico de pesquisa de interesse (PETERSEN *et al.*, 2015). A partir de um conjunto inicial de 1271 artigos, identificou-se 12 estudos primários para os quais tornou-se possível extrair cinco desafios (Colaboração, Qualidade, Incompletude, Manutenibilidade e Categorização) e três perspectivas gerais de soluções investigadas (Uso estratégico do README, Adoção de Inteligência Artificial e Ferramentas e Abordagens de Apoio).

O presente estudo apresenta perspectivas de contribuições para academia e prática. Do ponto de vista acadêmico, esta pesquisa oferece um MSL revelando um conjunto de desafios e soluções relacionadas à documentação de software, um tópico ainda pouco explorado no contexto de software de código aberto. Tal abordagem fornecerá uma visão abrangente da área, contribuindo para a consolidação do conhecimento e a identificação de lacunas de pesquisa. Do ponto de vista prático, este trabalho promove uma reflexão sobre o processo de documentação em projetos FOSS, reverberando desafios e soluções que podem contribuir para o aperfeiçoamento da qualidade das documentações e eficiência dos projetos.

1.1 Objetivos

O presente trabalho tem como objetivo geral mapear, com base nos últimos dez anos de pesquisa científica, os desafios e soluções em relação à documentação de software em projetos FOSS.

Já em relação aos objetivos específicos, pretende-se:

- Realizar uma caracterização dos estudos primários identificados;
- Identificar os desafios enfrentados na documentação de software em projetos FOSS;
- Identificar as soluções exploradas na documentação de software em projetos FOSS;

1.2 Estrutura do Trabalho

O presente trabalho está organizado em sete capítulos, além da presente Introdução, os quais são estruturados da seguinte forma:

- **Capítulo 2 - Background Teórico**: objetiva-se articular a fundamentação bibliográfica e os trabalhos relacionados envolvidos na compreensão dos elementos teóricos utilizados nesta pesquisa;
- **Capítulo 3 - Procedimentos Metodológicos**: descreve-se os aspectos relacionados ao método científico adotado nesta pesquisa;
- **Capítulo 4 - Resultados**: discute-se as análises e resultados desta pesquisa;
- **Capítulo 5 - Discussão**: busca-se apresentar uma discussão geral sobre os achados da pesquisa à luz da literatura;
- **Capítulo 6 - Ameaças à Validade**: discute-se as ameaças identificadas para os resultados deste trabalho e as estratégias adotadas para mitigar tais questões;
- **Capítulo 7 - Considerações Finais**: apresenta-se as contribuições e trabalhos futuros desta pesquisa.

2 BACKGROUND TEÓRICO

Esta seção articula a fundamentação bibliográfica e os trabalhos relacionados envolvidos na compreensão dos elementos teóricos utilizados nesta pesquisa. A princípio, na Seção 2.1, explana-se sobre sobre o processo de desenvolvimento de software em projetos de software livre e de código aberto (em inglês, *free and open-source software* ou FOSS). Posteriormente, na Seção 2.2, discute-se as principais definições de documentação de software sob a perspectiva da engenharia de software, além de ressaltar sua importância dentro do desenvolvimento de software FOSS. Dessa forma, torna-se possível apresentar uma visão geral da literatura, incluindo como o presente trabalho está inserido na mesma.

2.1 Desenvolvimento de Projetos FOSS

Projetos FOSS são fundamentais para o desenvolvimento de software moderno. Atualmente, muitos grandes sistemas de software dependem de estruturas e bibliotecas de código aberto (AVELINO *et al.*, 2019). De acordo com Angeren *et al.* (2011), o conceito geral por trás do software FOSS abrange artefatos de software, incluindo código-fonte, licenças, melhores práticas de desenvolvimento, inovação, ética, filosofia, movimento social, comunidade, cultura, governança e engajamento organizacional. Normalmente, os desenvolvedores são voluntários primários. Além disso, o software emerge de uma comunidade coordenada e não supervisionada de desenvolvedores e outros colaboradores. Tal processo de desenvolvimento é baseado no princípio de que os programas de software devem ser compartilhados livremente entre os usuários, dando-lhes a possibilidade de introduzir implementações e modificações (HUMES, 2007). Caracterizando-se de maneira diferente do desenvolvimento e distribuição do software tradicional, projetos de código aberto incluem a redistribuição gratuita, a inclusão do código-fonte e a possibilidade de modificações, que devem ser distribuídos nos mesmos termos do software original (PERENS *et al.*, 1999).

Segundo Krogh e Hippel (2006), o software FOSS é normalmente criado por um indivíduo ou um grupo de pessoas que deseja desenvolver um produto de software para atender às suas próprias necessidades. A primeira versão do código-fonte criado é disponibilizado gratuitamente à todos, permitindo que qualquer pessoa participe do processo do ciclo de desenvolvimento do software (WU; LIN, 2001). Stamelos *et al.* (2002) ressalta dois princípios que definem amplamente o poder do FOSS: (a) a evolução rápida para que muitos usuários/progra-

madores possam ter a oportunidade de usar o novo sistema e modificá-lo, e nenhum tempo é gasto em atividades de gerenciamento 'desnecessárias'; e (b) muitos programadores trabalhando ao mesmo tempo no mesmo problema, aumentando a probabilidade de sua solução.

A comunidade de código aberto é formada por desenvolvedores e usuários, que compartilham um interesse comum e interagem entre si para compartilhar conhecimento para o desenvolvimento de um projeto (KAUR *et al.*, 2022). As pessoas ingressam em comunidades FOSS em diferentes idades e têm diferentes origens, capacidades e recursos, bem como objetivos diferentes. Além disso, possuem diferentes níveis de habilidades de programação e experiência (LAKULU *et al.*, 2010). Segundo Steinmacher *et al.* (2014), o desenvolvimento de um projeto FOSS bem-sucedido depende essencialmente do trabalho de uma comunidade de desenvolvedores voluntários que estão distribuídos globalmente e colaboram através da Internet. Michlmayr *et al.* (2005), por sua vez, ressaltam que algumas das vantagens do modelo de desenvolvimento FOSS é o potencial para uma substancial revisão por pares e a atração de excelentes programadores de todo o mundo.

Projetos de código aberto podem ser complexos e podem envolver o processo de melhoria contínua. O processo de desenvolvimento de software é muito amplo e improvável para ser mantido por qualquer desenvolvedor de software ou por um pequeno grupo de desenvolvedores (DAI *et al.*, 2020). Para Jalote (2012), o processo de software envolve um conjunto de atividades que precisam estar interligadas por padrões e, se as atividades atuarem corretamente conforme os padrões, o resultado desejado é atingido. Contudo, as comunidades de código aberto são exemplos de conhecimento compartilhado, envolvendo ativos de diferentes indivíduos em um processo de interação e aprendizagem (ISKOUJINA; ROBERTS, 2015). O envolvimento ativo da comunidade é essencial para o sucesso de um projeto de código aberto de modo que as mesma deve se esforçar para obter maior participação e engajamento da comunidade por meio do uso de ferramentas, práticas e processos (KAUR *et al.*, 2022) Michlmayr *et al.* (2005) evidenciam três práticas de desenvolvimento e qualidade que são essenciais para uma boa estrutura de desenvolvimento de um projeto FOSS:

- **Infraestrutura:** Projetos FOSS dependem fortemente da infraestrutura que permite o desenvolvimento distribuído e a colaboração. Partes importantes da infraestrutura são: i) Sistemas de rastreamento de *bugs*: são usados para obter *feedback* dos usuários. Eles geralmente são usados para armazenar relatórios de *bugs* reais, bem como solicitações de recursos; ii) Sistemas de controle de versão: permitem que várias pessoas trabalhem

com o mesmo código simultaneamente e acompanhem quem faz quais alterações; iii) Compilações automáticas: garantem que o código mais recente no sistema de controle de versão ainda seja compilado e iv) Listas de discussão: usadas para a comunicação, entre ambos os desenvolvedores e usuários.

- **Processo:** O desenvolvimento FOSS segue muitos processos, mas um grande número deles não está documentado e os desenvolvedores aderem a eles implicitamente. Alguns dos processos são: i) Adesão: os projetos exigem que os possíveis membros sigam procedimentos específicos, em sua maioria não documentados, para ingressar em um projeto. Esses procedimentos variam consideravelmente entre os projetos. Alguns se certificam explicitamente de admitir apenas contribuidores de quem se pode esperar submissões de alta qualidade, enquanto outros projetos são mais liberais; ii) Revisão por pares: normalmente, as mudanças feitas no sistema de controle de versão são revisadas pelos membros dos projetos, na maioria dos casos essa forma de revisão por pares não é muito bem formalizada; iii) Testes: para garantir que uma nova versão atenda aos padrões de um projeto e que não tenha grandes regressões, alguns projetos possuem *checklists* de testes. Essas listas de verificação contêm as funções mais importantes e descrevem brevemente como elas podem ser testadas e iv) Garantia de qualidade: alguns projetos organizam dias de *bugs* ou festas de resolução de *bugs* para fazer a triagem de seus *bugs* pendentes. Durante este trabalho, relatórios de *bugs* duplicados são marcados como tal, *bugs* antigos são reproduzidos e *bugs* também são corrigidos.
- **Documentação:** Em projetos FOSS a documentação referente às práticas de desenvolvimento deve ser explícita, também documentações descrevendo as formas de contribuir e como ingressar no projeto. Dois tipos de documentações são: i) Estilos de codificação: documentação destinada a desenvolvedores que descreve o estilo que deve ser usado para o código-fonte e ii) *Code commit*: documentação que descreve quando e quem pode fazer alterações no sistema de controle de versão de um projeto.

Nesse contexto, esta é apenas uma visão parcial do que realmente deve constituir (e avançar) um projeto FOSS, que geralmente se baseia em uma comunidade de usuários com uma rica variedade de perfis. Assim, todos são convidados a ajudar mesmo que não saibam escrever código, auxiliando assim na sustentabilidade dos projetos FOSS (IZQUIERDO; CABOT, 2022). Em específico, Michlmayr *et al.* (2007) ressaltam que o código aberto é caracterizado por um modelo de desenvolvimento altamente iterativo no qual novos lançamentos de desenvolvimento

são normalmente disponibilizados com muita frequência. O uso de ferramentas como sistemas de controle de versão e rastreamento de *bugs* serve funções importantes durante o gerenciamento de *releases*, pois fornecem uma boa visão geral do status do projeto.

Em suma, os projetos de desenvolvimento FOSS muitas vezes dependem de uma autoatribuição de tarefas com pouca coordenação, mas todos os desenvolvedores precisam alinhar suas atividades em direção a um objetivo comum durante o ciclo de desenvolvimento (MICHLMAYR *et al.*, 2007). Assim, coordenar e organizar o trabalho em projetos FOSS, portanto, envolve combinar a demanda de esforço (recursos desejados e *bugs* conhecidos que levarão tempo e habilidades especializadas para consertar) com oferta de esforço (voluntários e desenvolvedores pagos que têm suas próprias motivações e prioridades) (KLUG *et al.*, 2021).

2.2 Documentação de Software

A documentação de software pode ser definida como um artefato cuja finalidade seja comunicar a informação sobre o sistema de software ao qual ele pertence (FORWARD, 2002). Ou seja, a documentação estabelece uma comunicação entre todos os envolvidos e nivela o conhecimento destes sobre o projeto, objetivando transferir e compartilhar conhecimento (SILVA, 2020). A documentação de software é considerada parte integrante do processo de desenvolvimento de software e, consequentemente tem vários usos no ciclo de vida do software, por exemplo, sendo usada como um meio de comunicação entre as partes interessadas, sendo um repositório de informações para o manter o software atualizado e, finalmente, como um guia para novos usuários do projeto (DING *et al.*, 2014b). Dessa forma, a documentação consiste em um conjunto de manuais gerais e técnicos, podendo ser organizado em forma de textos e comentários, utilizando ferramentas do tipo dicionários, diagramas e fluxogramas, gráficos, desenhos, dentre outros (COELHO, 2009).

Como evidencia Chapin (2000), o valor do software pode ser determinado por muitos fatores, dentre os quais pode-se destacar a documentação, incluindo sua preparação e manutenção. Além disso, é importante se atentar a qualidade, obsolescência ou falta de conteúdo na documentação. Contudo, a construção da documentação qualificada não é trivial e a diversidade de modelos de documentos existentes também ocasiona dúvidas (SILVA, 2020). Coelho (2009) destaca que o desenvolvedor deve encontrar ferramentas que facilite o entendimento da documentação, tanto para o administrador quanto para o futuro usuário do software. Parnas (2010) reforça que um documento de software precisa ser correto e preciso, ou seja, espera-se que as

informações que alguém pode obter do documento sejam realmente verdadeiras para o sistema que está sendo descrito, não havendo dúvidas sobre o que eles significam.

Segundo Michelazzo (2008) a documentação no ciclo de desenvolvimento do software pode ser dividida em dois grandes grupos: o primeiro grupo é a *Documentação Técnica*, considerada mais simples, pois descreve o trabalho do desenvolvedor. Sendo voltada para o uso do desenvolvedor, a *Documentação Técnica* compreende a dicionários e modelos de dados, fluxogramas de processos e regras de negócios, dicionários de funções e comentários de códigos. O segundo grupo é a *Documentação de Uso* e é considerada mais complexa, pois requer habilidades especiais para a redação de manuais, inserção de *screenshots*, desenhos e outros elementos gráficos. A *Documentação de Uso* é voltada para o usuário final e administrador do sistema, sendo formada por apostilas ou manuais que apresentam como o sistema deve ser usado, o que esperar dele e como receber as informações que se deseja (AGHAJANI *et al.*, 2020).

Garousi (2012) reforça que quando um documento é criado, ele pode ser usado como um relatório tutorial ou como um documento de referência. Para Cioch *et al.* (1996), existem quatro estágios de experiência (de recém-chegado, o primeiro dia de trabalho; a especialista, após alguns anos de trabalho em um sistema). Para cada etapa, eles propõem diferentes documentos: os recém-chegados precisam de uma breve visão geral do sistema; os aprendizes precisam da arquitetura do sistema; os estagiários precisam de documentos orientados a tarefas, como descrição de requisitos, descrição de processos, exemplos, instruções passo a passo; finalmente, os especialistas precisam de documentação de baixo nível, bem como descrição de requisitos e especificação de projeto.

Dessa forma, Ambler (2001) justifica que há duas razões básicas para documentar: auxiliar a comunicação durante o desenvolvimento do software e auxiliar o entendimento nas atividades de manutenção e atualização quando se fizerem necessárias. Michelazzo (2008) descreve quatro tipos de documentação de software mais usuais:

- **Documentação de código:** feita basicamente de duas formas – comentários dentro do próprio código e geração de documentação online. O desenvolvedor deve ter a consciência de que não será o único a "colocar a mão no sistema" e por isso deve fazer o comentário dos códigos de maneira bastante clara.
- **Modelos de dados:** refletem de forma gráfica (e lógica) a base de dados de um sistema, seus relacionamentos, entidades, chaves e tudo aquilo que se refere aos dados em si. É, portanto, considerado peça fundamental para o desenvolvimento de um sistema e por isso

são pensados e criados antes do início do desenvolvimento. Podem ser criados por meio de engenharia reversa ou ainda baseando-se nas necessidades do aplicativo que está sendo envolvido.

- **Dicionários de dados:** arquivo ou documento que define a organização básica dos dados do banco. Nele são informadas as tabelas, os campos, suas definições, tipos e descrições.
- **Fluxogramas:** apresentam graficamente a sequencia lógica das informações de um processo ou sistema, utilizando vários elementos de geometrias diferentes que indicam uma das partes do processo. Visualmente conseguem passar a lógica de todo um sistema desde os níveis mais altos de processamento até pequenas partes, permitindo, assim, uma visão geral do que realmente precisa ser feito dentro de um sistema.

Em consonância com Michelazzo (2008), Ambler (2001) salientam que a documentação de software responde a três necessidades: (i) contratual; (ii) apoiar um projeto de desenvolvimento de software permitindo que os membros da equipe concebam gradualmente a solução a ser implementada e (iii) permitir que uma equipe de desenvolvimento de software comunique os detalhes da implementação ao longo do tempo para a equipe de manutenção. Portanto, a documentação de software é essencial para projetos *open source*, pois ela segmenta dois princípios básicos do software livre: produção colaborativa e divulgação ampla. Sempre que existe mais de uma pessoa desenvolvendo um trabalho (software ou qualquer outro), a comunicação é fundamental. Existem diversas maneiras de trabalhar em conjunto, mas quando o objeto final não pode ser segmentado para que cada um realize uma tarefa individual sem que ela afete o conjunto (o que talvez seja mesmo impossível), registrar e disponibilizar mudanças, coisas por fazer, decisões, discussões e o próprio objeto do trabalho é crucial (MATTE, 2008).

Sendo projetos que possuem conteúdo aberto e colaborativo, a comunidade de código aberto busca manter as documentações atualizadas. Nesse sentido, projetos FOSS possuem características singulares quanto a sua organização, principalmente porque esses projetos precisam lidar com comunidades de desenvolvedores (PINTO, 2021). Winters *et al.* (2020) salientam, por exemplo, que o *onboarding* de novo membros de uma equipe ou base de código exige muito menos esforço se o processo for claramente documentado. Em projetos FOSS, os contribuidores são mais focados quando as metas de design e objetivos do projeto são claramente declarados. Em suma, a documentação de software se mostra um recurso inestimável para qualquer projeto de software, pois ajuda as partes interessadas a usar, entender, manter e desenvolver um sistema (AGHAJANI *et al.*, 2020).

Durante a análise da literatura, também verificou-se estudos secundários que se preocupavam com documentação de software. No entanto, nenhum focou explicitamente no contexto dos desafios e soluções enfrentados pelo software livre. Zhi *et al.* (2015) empregaram uma metodologia de Mapeamento Sistemático da Literatura (MSL) para mapear o conhecimento existente sobre custo, benefício e qualidade da documentação de software. Descobriram que o aspecto do custo da documentação parece ter sido negligenciado na literatura existente e não existem métodos ou modelos sistemáticos para medir a relação de custo-beneficio. Além disso, Theunissen *et al.* (2022) realizaram um MSL para identificar e analisar pesquisas sobre documentação em Desenvolvimento Contínuo de Software. Eles observaram desafios como a compreensão da documentação informal, a percepção da documentação como desperdício, a medição da produtividade baseada apenas em software funcional, inconsistências entre documentação e software e um foco no curto prazo. Além disso, identificaram práticas relacionadas à comunicação não escrita e informal, utilizando artefatos de desenvolvimento para documentação e adotando *frameworks* de arquitetura.

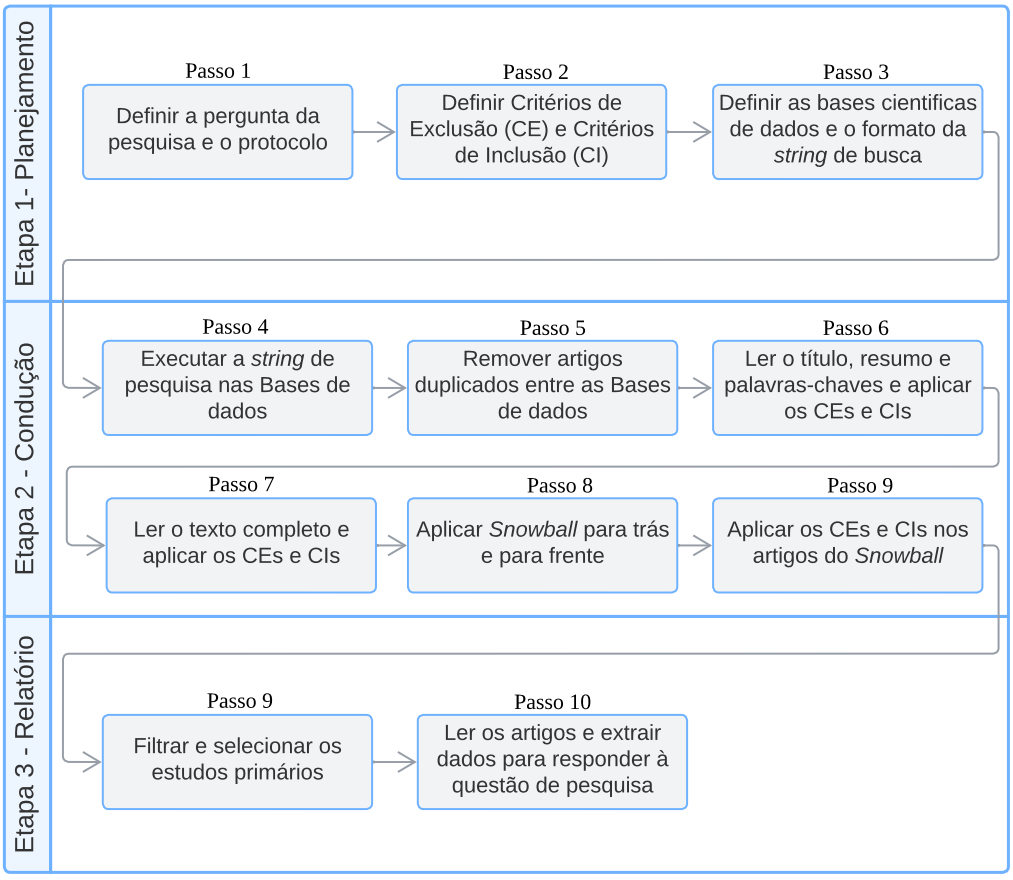
Já Habib e Romli (2021) conduziram um MSL focado em explorar os problemas e a importância da documentação no desenvolvimento ágil. A partir deste estudo, os autores identificaram 14 aspectos relacionados à documentação em metodologias ágeis que foram investigados, destacando a importância de ter documentação "suficiente" em metodologias ágeis. Ainda no contexto do desenvolvimento ágil de software, Islam *et al.* (2023) conduziram uma Revisão Sistemática da Literatura (RSL) com o objetivo de identificar sistematicamente o que documentar, quais ferramentas e métodos de documentação estão em uso e como essas ferramentas podem superar os desafios de documentação. Ding *et al.* (2014c) também empregaram um método RSL para identificar estudos primários sobre abordagens baseadas em conhecimento em documentação de software. Eles observaram que a compreensão da arquitetura é o principal benefício do uso de abordagens baseadas em conhecimento na documentação de software. No entanto, também observaram que o custo da recuperação de informações de documentos continua a ser uma grande preocupação quando se empregam abordagens baseadas no conhecimento na documentação.

3 PROCEDIMENTOS METODOLÓGICOS

O percurso metodológico trilhado neste trabalho apoia-se num Mapeamento Sistemático da Literatura (MSL) com o objetivo de mapear, com base nos últimos dez anos de pesquisa científica, os desafios e soluções em relação à documentação de software em projetos de código aberto. Diferentemente da Revisão Sistemática da Literatura que envolve uma avaliação crítica rigorosa dos estudos selecionados, o MSL é mais exploratório, visando mapear os estudos existentes para fornecer uma visão geral do campo de pesquisa (KITCHENHAM *et al.*, 2007).

A Figura 1 apresenta a visão geral do plano metodológico adotado nesta pesquisa. Na **Etapa 1 (Planejamento)**, inicialmente, realizou-se manualmente pesquisas exploratórias usando o *Google Scholar* para refinar os critérios de inclusão e exclusão, questão de pesquisa, sinônimos para a *string* de pesquisa e modelo de extração de informações. Para definição do protocolo de pesquisa, o presente trabalho se inspirou no MSL de Trinkenreich (2021), bem como seguiu as diretrizes especificas de Petersen *et al.* (2008) e Kitchenham *et al.* (2007).

Figura 1 – Procedimentos metodológicos para execução do MSL.



Fonte: Adaptado de (KITCHENHAM *et al.*, 2007).

No protocolo, visando a seleção de trabalhos que abordam explicitamente o tema documentação de software em projetos de software livre e de código aberto (em inglês, free and open-source software ou FOSS), definiu-se diferentes Critérios de Inclusão (CI) e Critérios de Exclusão (CE) para a filtragem dos artigos. Se um artigo fosse condizente a um CE, logo o artigo deveria ser removido do estudo. Adicionalmente, um artigo só poderia ser mantido se atendesse todos CI. Desse modo, os CIs e CEs definidos para a triagem dos trabalhos foram, respectivamente:

- **(+) CI1:** Trabalhos que contém as palavras-chave da expressão de busca, no resumo e/ou título e/ou nas palavras-chave do artigo selecionado;
- **(+) CI2:** Artigos publicados em um local revisado por pares, incluindo *workshops*, conferências, simpósios e periódicos;
- **(+) CI3:** Artigos onde o idioma é o inglês.
- **(-) CE1:** Artigos não disponíveis em texto completo;
- **(-) CE2:** Artigo que não contém resumo;
- **(-) CE3:** Artigos onde documentação de software são usadas apenas como exemplo;
- **(-) CE4:** Artigos secundários ou terciários (por exemplo, revisões sistemáticas da literatura, pesquisas, etc).
- **(-) CE5:** Artigos na forma de tutorial, editoriais e etc., porque não fornecem informações suficientes;

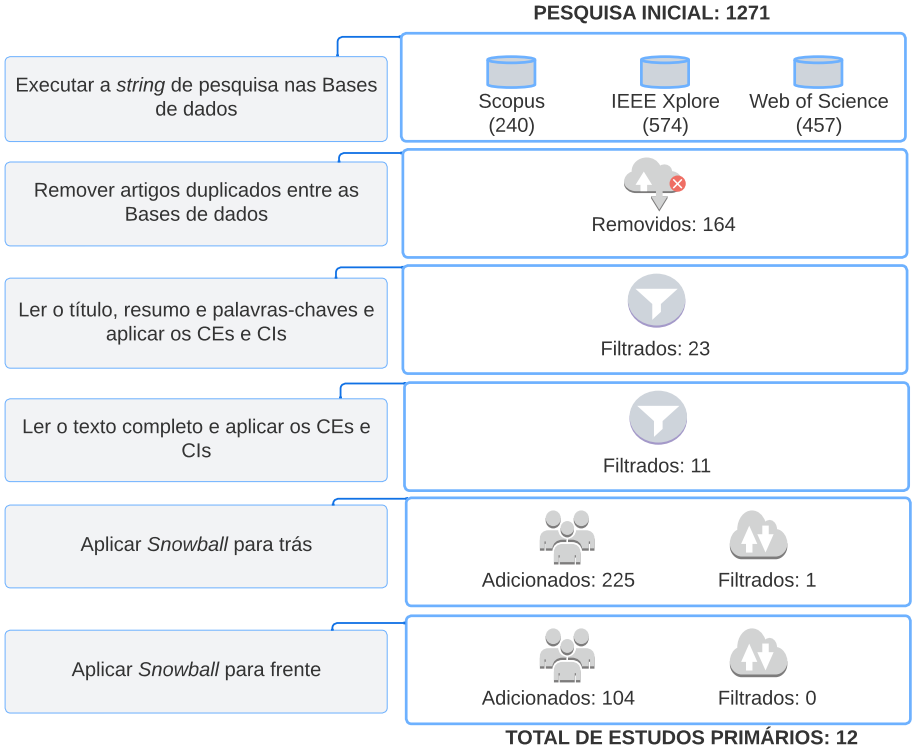
Ademais, as bases de dados selecionadas para a pesquisa foram *IEEE Xplore* , *Web of Science* e *Scopus*. Tais bibliotecas foram selecionadas porque (i) indexam sites relevantes para este estudo, (ii) suportam pesquisas usando uma expressão booleana, (iii) fornecem acesso a textos completos e (iv) permitem a exportação de resultados devidamente formatados. A *string* de busca foi definida com termos relevantes ao foco da pesquisa, sendo ajustada conforme necessário: "(`open source` OR `open-source` OR `OSS` OR `free software` OR `FOSS OR FLOSS`) **AND** (`software development` OR `software engineering`) **AND** (`documentation` OR `document` OR `guide` OR `writing` OR `artifact` OR `specification` OR `notes` OR `knowledge base` OR `report` OR `whitepaper`)". Tal *string* de busca combina vários termos e sinônimos relacionados ao desenvolvimento de software *open source* e documentação de software. Nesse caso, abrange-se diferentes variações, como *"open source"* e *"open-source"*, bem como acrônimos como "OSS"e "FOSS". Além disso, inclui-se termos relacionados à documentação ou documentos, vide *guide*, *writing*, *artifact*, *specification*, *notes*, *knowledge base*,

report e *whitepaper*. Considerou-se os trabalhos que abordam explicitamente a documentação de software em projetos de código aberto e cujo método de pesquisa é de base empírica. A busca foi realizada em julho e agosto de 2023.

Desse modo, na **Etapa 2 (Condução)**, executou-se a *string* de busca nas bases de dados selecionadas onde foram retornados 574 artigos no *IEEE Xplore*, 457 artigos no *Web of Science* e 240 artigos no *Scopus*, totalizando inicialmente 1271 artigos. Os artigos filtrados foram organizados em uma planilha no *Google Sheets* para se ter uma melhor visualização e organização dos dados. Primeiramente, foram removidos os artigos duplicados entre as bases de dados. Assim, 164 artigos duplicados foram removidos, restando 1107 artigos para serem analisados. Em seguida, os artigos filtrados foram divididos em dois grupos e dois pesquisadores convidados aplicaram os CEs e CIs (uma pessoa por grupo), enquanto o autor principal realizou uma dupla verificação em todos os artigos (ambos os grupos). Os pesquisadores se reuniram virtualmente em conflitos de decisão e chegaram a um consenso de seleção preliminar. Outros dois autores mais experientes foram convidados para discutir a decisão final caso o conflito permanecesse. Assim, além do autor principal, outras quatro pessoas estiveram envolvidas no processo de filtragem dos artigos, reforçando o rigor do protocolo de pesquisa explorado.

Durante as discussões relacionadas a filtragem dos artigos, os pesquisadores utilizaram uma abordagem de tomada de decisão dialética onde cada pesquisador defendeu a manutenção ou a retirada do artigo, e então um consenso foi alcançado após os debates (SCHWEIGER *et al.*, 1986). Essa abordagem ajudou a avaliar sistematicamente os argumentos para manter ou remover um artigo e evitar o viés de tomada de decisão de confiar fortemente na impressão inicial sobre um artigo. Após aplicar os ECs e ICs, reduziu-se para um conjunto de 11 artigos. A partir da lista de 11 artigos, aplicou-se a abordagem interativa de *Snowballing Backward* de forma independente para complementar os artigos filtrados no banco de dados. Nesse sentido, excluiu-se os trabalhos duplicados e os que não atenderam aos ECs e ICs. Como resultado, incluiu-se mais um artigo aos estudos primários, totalizando uma lista final de 12 artigos primários. Também conduzimos o processo *Snowball Forward* analisando as citações de nossos estudos primários, mas nenhum novo estudo primário foi encontrado de acordo com os critérios de inclusão e exclusão. A Figura 2 sintetiza o processo de seleção passo a passo discutido previamente.

Figura 2 – Passo a passo da seleção dos estudos primários.



Fonte: Adaptado de (TRINKENREICH, 2021).

Finalmente, na **Etapa 3 (Relatório)**, organizou-se a filtragem final dos estudos primários para fins de geração do relatório analítico. Ao todo, foram selecionados 12 artigos. Os artigos foram organizados em uma planilha no *Google Sheets* tendo como campos de dados padrão: Título do trabalho, nome dos autores, país, palavras-chave, ano de publicação, tipo de publicação, local de publicação, base de dados, desafios identificados e soluções exploradas. A Tabela 1 sintetiza os 12 artigos que formam as fontes primárias para o estudo. Todos os dados da pesquisa estão disponíveis no repositório de suporte da pesquisa (PINHO, 2024), promovendo assim transparência e acessibilidade aos processos explorados em nosso MSL.

Uma perspectiva quali-quantitativa foi explorada para a análise dos dados. Quanto à análise quantitativa, utilizou-se estatística descritiva para analisar dados sobre distribuição por ano de publicação, estudos por bases de dados e locais de publicação. Quanto à perspectiva qualitativa, a codificação aberta foi explorada com o objetivo de categorizar e obter dados relevantes dos estudos para responder à nossa questão de pesquisa (SALDAÑA, 2021). Inicialmente, o primeiro autor conduziu a codificação de forma independente, derivando-a dos trechos dos artigos. Esses códigos pós-formados foram então associados e categorizados seguindo uma cadeia de evidências. Para garantir a confiabilidade em nossa análise qualitativa, dois coautores mais experientes e com

expertise em pesquisa em Engenharia de Software revisaram e discutiram os códigos até chegar a um consenso. Foram realizadas quatro reuniões entre esses autores para refinar os códigos e aumentar a confiabilidade dos resultados. Todas as etapas do estudo, incluindo os códigos extraídos, estão documentadas no repositório da pesquisa (PINHO, 2024).

Tabela 1 – Síntese dos estudos primários selecionados.

Id	**References**	**Publication Venue**
PS01	Carvalho *et al.* (2014)	Computer Science and Information Systems
PS02	Ding *et al.* (2014a)	International Conference on Engineering of Complex Computer Systems
PS03	Bigliardi *et al.* (2014)	International Conference on Quality Software
PS04	Aversano *et al.* (2017)	International Conference on Evaluation of Novel Approaches to Software Engineering
PS05	Ma *et al.* (2018)	International Conference on Mining Software Repositories
PS06	Prana *et al.* (2019)	Empirical Software Engineering
PS07	AlOmar *et al.* (2021)	Journal of Software: Evolution and Process
PS08	Pasuksmit *et al.* (2022)	International Conference on Mining Software Repositories
PS09	Puhlfürß *et al.* (2022)	International Conference on Software Maintenance and Evolution
PS10	Sun *et al.* (2023)	International Conference on Mining Software Repositories
PS11	Wermke *et al.* (2023)	Symposium on Security and Privacy
PS12	Ciurumelea *et al.* (2023)	Empirical Software Engineering

Fonte: Autoria própria.

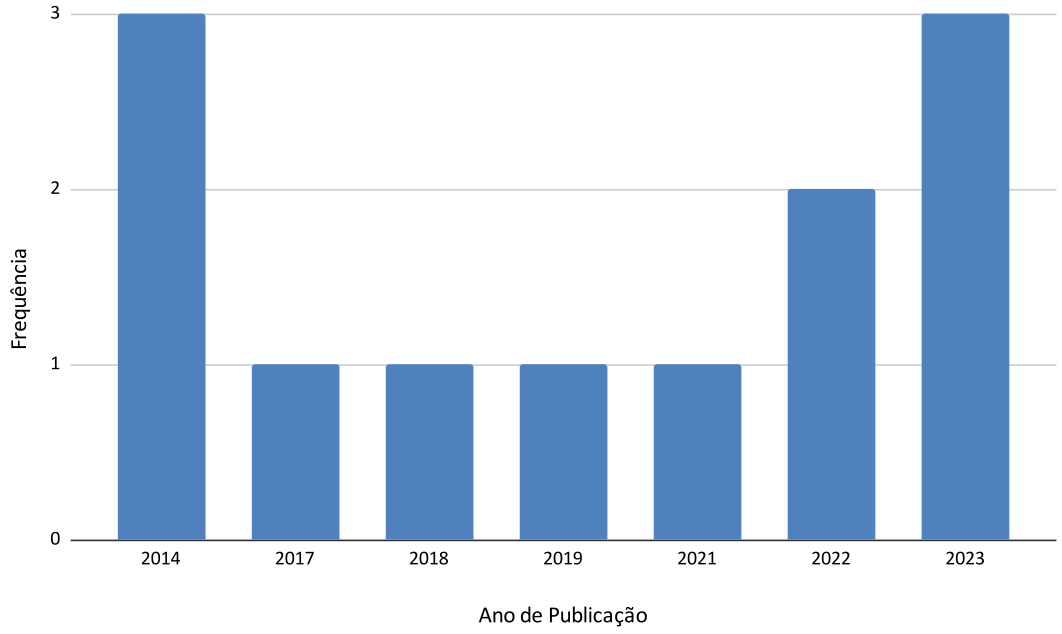
4 RESULTADOS

Nesta seção, apresentamos nossas análises em três seções principais. Primeiramente, discutimos sobre a caracterização dos estudos primários, explanando sobre a distribuição de publicações por ano, país dos autores, fontes de publicação, locais de publicação e método de pesquisa abordado pelos artigos. Posteriormente, discutimos em profundidade as perspectivas que emergiram da nossa análise em relação aos desafios identificados nas documentações de software em projetos de software livre e de código aberto (em inglês, *free and open-source software* ou FOSS) e, finalmente, as soluções exploradas para mitigar os desafios nas documentações de software em projetos FOSS.

4.1 Caracterização dos Estudos Primários

A Figura 3 mostra a distribuição de estudos primários por ano. O período de filtragem foi do ano de 2013 ao ano de 2023. Há um intervalo de quatro anos sem publicações entre os anos de 2013 e 2023. Os artigos mais antigos foram publicados no ano de 2014. Desde 2021, podemos observar que pelo menos um artigo foi publicado. Os anos de 2014 e 2023 foram os anos com mais artigos publicados, ambos com três artigos.

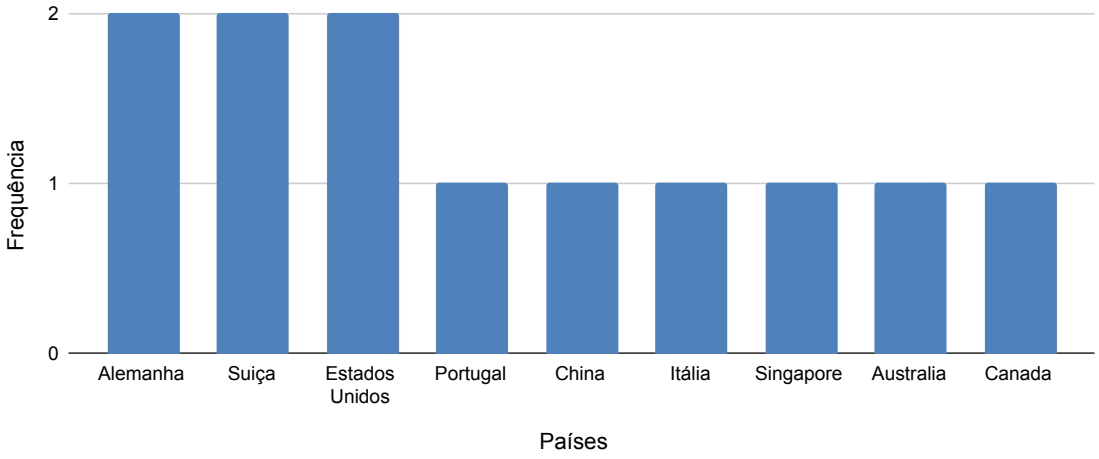
Figura 3 – Ano de publicação dos artigos primários selecionados.



Fonte: Autoria própria.

A distribuição geográfica dos países pertencentes aos primeiros autores é mostrada na Figura 4. Consideramos para está análise o país em que o primeiro autor pertence. Os Estados Unidos, Suíça e Alemanha tiveram a mesma quantidade de registro de artigos (2), sendo mais predominante países do continente Europeu.

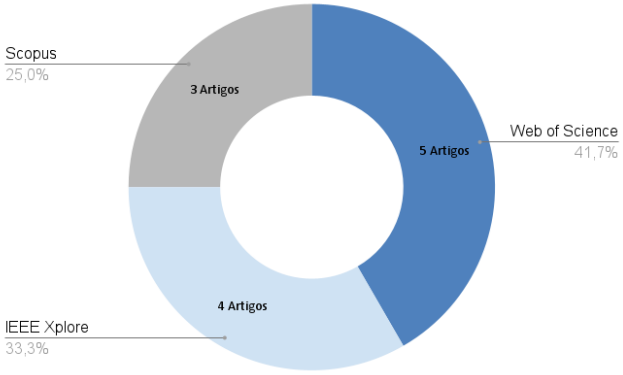
Figura 4 – Países dos primeiros autores.



Fonte: Autoria própria.

A Figura 5 mostra os artigos selecionados por bases de dados e os locais de publicação. *Web of Science* foi a base de dados que teve um maior número de artigos selecionados 41,7% (5 artigos), seguido do *IEEE Xplore* com 33,3% (4 artigos) e *Scopus* com 25,0% (3 artigos). Observando os locais de publicação, três dos doze estudos primários selecionados foram publicados na *International Conference on Mining Software Repositories (MSR)*, os demais se subdividem entre áreas e eventos diferentes. Além disso, os resultados mostraram que 66,7% (8 artigos) dos artigos foram publicados em Conferências.

Figura 5 – Bases de dados e locais de publicação.



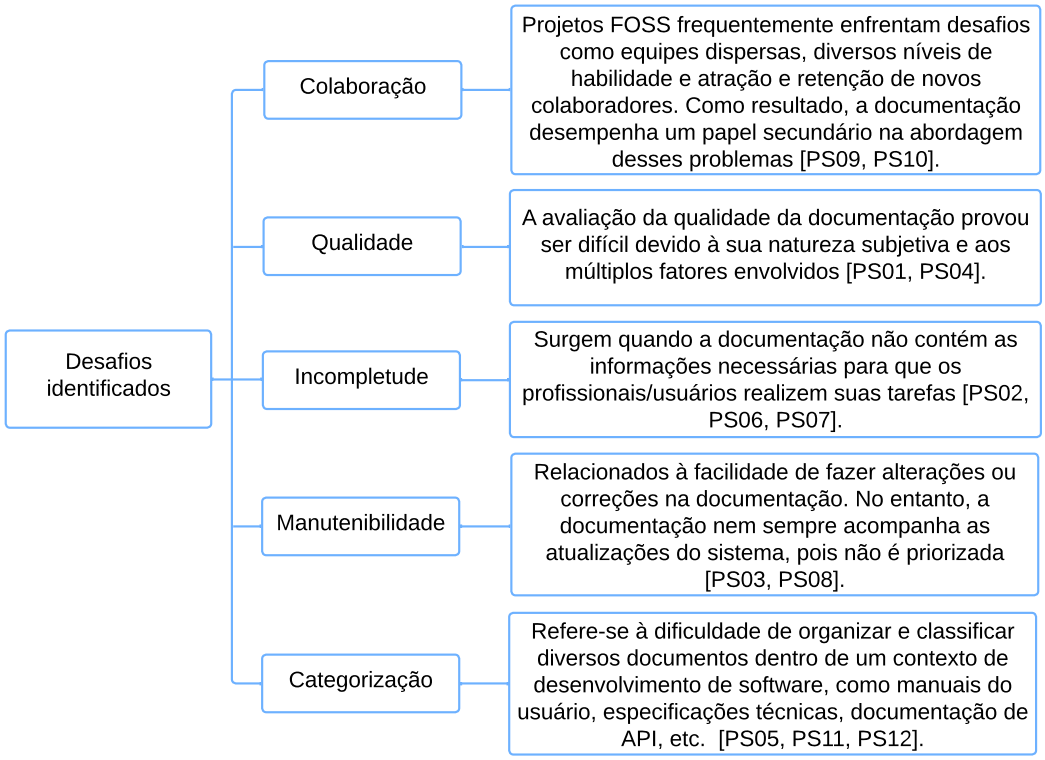
Publicação Venue	Conference	Journal
Computer Science and Information Systems		1
International Conference on Engineering of Complex Computer Systems		1
International Conference on Quality Software		1
International Conference on Evaluation of Novel Approaches to Software Engineering		1
International Conference on Mining Software Repositories		3
Empirical Software Engineering		2
Journal of Software: Evolution and Process		1
International Conference on Software Maintenance and Evolution (ICSME)		1
Symposium on Security and Privacy (SP)		1

Fonte: Autoria própria.

4.2 Desafios Identificados

A literatura apresenta diferentes perspectivas em relação ao uso das documentações de software em projetos de código aberto que culmina em uma série de desafios enfrentados pelos *stakeholders*. Em teoria, uma boa documentação é um recurso inestimável para qualquer projeto de software, pois ajuda as partes interessadas a usar, entender, manter e desenvolver um sistema (AGHAJANI *et al.*, 2020). Assim, sabendo que a documentação ajuda os engenheiros de software a realizar tarefas de manutenção e desenvolvimento com mais eficiência, faz-se necessário compreender quais as atenuações para se ter documentações úteis e de qualidade (GAROUSI *et al.*, 2013). A Figura 6 sintetiza os desafios da documentação de software FOSS extraídos e categorizados de nossos estudos primários, nomeadamente Colaboração, Qualidade, Incompletude, Capacidade de Manutenção e Categorização. Esses desafios são discutidos a seguir.

Figura 6 – Desafios na documentação de projetos FOSS.



Fonte: Autoria própria.

A **colaboração** na documentação de software para projetos FOSS encontra desafios de equipes distribuídas, dificultando a comunicação em tempo real e causando atrasos (ABER-

DOUR, 2007; CROWSTON *et al.*, 2007). As inconsistências surgem de diferentes níveis de especialização, prioridades conflitantes e o desenvolvimento dinâmico exige revisão contínua. Sun *et al.* (2023) [PS10] afirmam que uma melhor compreensão da colaboração entre desenvolvedores pode resultar no desenvolvimento de melhores artefatos colaborativos. Mais do que identificar padrões colaborativos em projetos FOSS, os autores destacaram que o tipo de arquivo mais colaborativo é os arquivos de testes, a documentação vem em segundo plano em relação ao trabalho conjunto. A justificativa é que os arquivos de testes tem um envolvimento de múltiplas partes interessadas e fornecem um *feedback* imediato. Em outra perspectiva, Puhlfürß *et al.* (2022) [PS09] explana que alguns projetos populares usam o *GitHub* principalmente para tornar seu código público e não para desenvolver uma comunidade de colaboradores que priorizaria uma boa documentação disponível publicamente.

A **qualidade** da documentação de software é um desafio multifacetado. No entanto, para impactar efetivamente o desenvolvimento de software, a garantia de qualidade também deve abordar sistematicamente a qualidade da documentação de software (PLÖSCH *et al.*, 2014). Aversano *et al.* (2017) [PS04] realizaram um estudo de caso objetivando avaliar a qualidade da documentação em sistemas de código aberto para compreender o suporte que ela pode oferecer. Segundo os autores, os resultados indicam que a documentação nem sempre está disponível e atende apenas parcialmente às necessidades dos desenvolvedores, pois muitas vezes está errada, incompleta, desatualizada e ambígua. Por outro lado, Carvalho *et al.* (2014) [PS01] fornecem em sua pesquisa uma visão sobre a dificuldade de avaliar a qualidade das documentações, devido a sua subjetividade. Assim, descrevem três características que impacta diretamente na qualidade geral das documentações, são elas i) Legibilidade: a legibilidade do texto pode ser subjetiva, mas existem características linguísticas que geralmente dificultam a leitura. Alguns deles podem até ser medidos, como por exemplo, o número de erros de sintaxe ou o uso excessivo de abreviaturas, ii) Atualidade: esta é uma característica importante da documentação e de outros arquivos textuais, eles deve estar atualizado e consultar a versão mais recente do software e iii) Completude: esta característica nos diz o quanto a documentação está completa e se aborda todos os tópicos necessários.

A **incompletude** das documentações surge se a documentação não contém as informações sobre o sistema ou seus módulos necessários para os profissionais/usuários para executar suas tarefas (ZHI *et al.*, 2015). Ding *et al.* (2014a) [PS02] argumentam que em vez de coletar e documentar informações de projetos FOSS, os desenvolvedores preferem usar fóruns, listas de

discussão e mídias sociais para obter informações e, assim, despender menos esforço. Segundo os autores, a documentação de arquitetura não é amplamente criada e utilizada em projetos FOSS. Além disso, a documentação relacionada à refatoração também é incomum em software de código aberto. Porém, de acordo com AlOmar *et al.* (2021) [PS07] fornecer documentação suficiente relacionado às refatorações realizadas no sistema é importante para facilitar o processo de revisão de código. Mas a falta de documentação de refatoração é um desafios que impacta a eficiência do processo de revisão de software de código aberto. Com outro estudo, Prana *et al.* (2019) [PS06] introduz que os arquivos *README* no *GitHub* desempenham um papel essencial na formação da primeira impressão que um desenvolvedor tem de um repositório de software e na documentação do projeto de software que o repositório hospeda. No entanto, falta-nos uma compreensão sistemática do conteúdo de um arquivo *README* típico, muitos arquivos *README* carecem de informações sobre a finalidade e o status de um repositório.

Além disso, a **manutenibilidade** da documentação diz respeito a questões relacionadas à facilidade de aplicação de alterações ou correções à mesma (AGHAJANI *et al.*, 2020). Bigliardi *et al.* (2014) [PS11] afirmam que cada projeto de código aberto tem suas peculiaridades, mas alguns mostram uma correlação positiva com o esforço de manutenção à medida que o sistema evolui, enquanto outros mostram o comportamento oposto. Do ponto de vista quantitativo, os autores observaram que uma parcela significativa dos artefatos não relacionados ao código-fonte não evolui junto com o projeto FOSS. Pasuksmit *et al.* (2022) [PS08], por sua vez, trazendo enfoques com base no manifesto ágil, explana que a documentação abrangente tem menor prioridade do que o software funcional. Então, é possível que a equipe ágil não promova informações adequadas ou atualizadas na documentação quando for necessário para estimativa de esforço e planejamento da *sprint*.

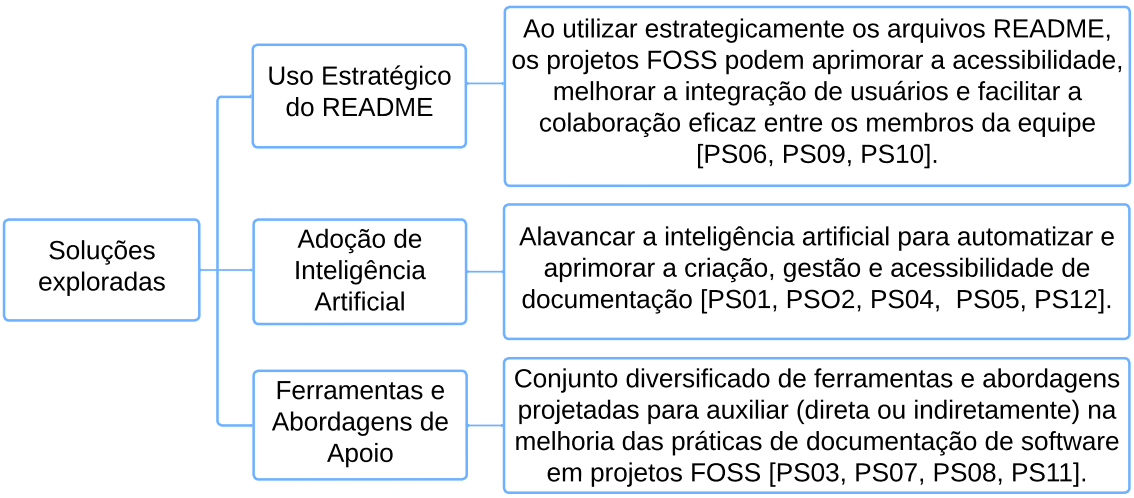
Por fim, **categorizar** diferentes tipos de documentação revelou-se um desafio, conforme descrito na literatura existente. Mais especificamente , este desafio refere-se à dificuldade em organizar e classificar vários documentos dentro de um contexto de desenvolvimento de software, tais como manuais de usuário, especificações técnicas, documentação de APIs, etc (AGHAJANI *et al.*, 2019). Ma *et al.* (2018) [PS05] investiga como os artefatos podem ser categorizados e que tipos de artefatos são criados durante o desenvolvimento de software FOSS. Os autores frisam que categorizar as documentações pode fornecer *insights* sobre projetos de software do ponto de vista técnico e de gerenciamento. Além disso, Wermke *et al.* (2023) [PS11] destacou que as perspectivas sobre o fornecimento de documentação parecem depender do con-

texto da equipe, o suporte as documentações aparentemente estão correlacionadas ao tamanho da equipe e o número de desenvolvedores menos experientes. Sendo importante seguir um estilo de formatação especifico e de estrutura bem definida, os comentários de código também seguem como um tipo de documentação que pode trazer um apoio significativo aos desenvolvedores (CIURUMELEA *et al.*, 2023) [PS12].

4.3 Soluções Exploradas

De acordo com a análise dos estudos primários, fomos capazes de observar caminhos estratégicos que podem contribuir para melhorar os desafios encontrados nas documentações de software em projetos FOSS. As comunidades de código aberto precisam de uma visão concisa dos diferentes tipos de ações para selecionar aquelas que são viáveis e adequadas às suas necessidades e aos desafios que enfrentam (TRINKENREICH, 2021). Resumimos as estratégias propostas na literatura na Figura 7. A seguir, explicamos cada uma delas, sejam elas processos, ferramentas ou práticas.

Figura 7 – Soluções aplicadas na documentação de software FOSS.



Fonte: Autoria própria.

O **Uso Estratégico do README** em projetos FOSS surgiu como uma solução potencial que merece destaque. Esses arquivos são frequentemente incluídos para fornecer informações essenciais sobre o projeto, como instruções de instalação, diretrizes de uso e

recursos principais (KOSKELA *et al.*, 2018). Ao utilizar arquivos *README* estrategicamente, os projetos de software podem melhorar a acessibilidade, melhorar a integração do usuário e facilitar a colaboração eficaz entre os membros da equipe (LIU *et al.*, 2022). Sun *et al.* (2023) [PS10] investigam a frequência de colaboração em diferentes tipos de arquivos em projetos de código aberto usando o *Author Cross Entropy* (ACE) e aproveitando o conjunto de dados do *World of Code* (WoC) e da API *GitHub*. Como os tipos de arquivo têm diferentes tamanhos e particularidades, foi usada a classificação *U de Mann-Whitney* para comparar a distribuição ACE em cada tipo de arquivo. A partir dos resultados, os arquivos de teste relatam um grau mais alto de colaboração entre os desenvolvedores e em seguida as documentações. Segundo os autores, a colaboração em documentações na plataforma *GitHub* é um viés positivo, pois embora esteja em um nível intermediário, é possível analisar um grau de versalidade de *commits* de alteração nas documentações. Além disso, é observado que para reforçar a colaboração com colegas de equipe, uma vez que os autores façam modificações na documentação, eles geralmente adicionam comentários para explicar suas mudanças estimulando também a interação. Puhlfürß *et al.* (2022) [PS09] por meio de uma análise de conteúdo exploratória, destaca que o *README* no *GitHub* quando usado de forma estruturada facilita a colaboração. Os contribuidores preferem colocar toda a documentação em arquivos *README* se for necessário documentar muitas páginas. Os recursos do *GitHub*, como rastreador de problemas, *Wiki* e solicitações *Pull* são usados para criar conexões subsequentes no projeto. Outro fator é que os projetos estudados usavam combinações de vários tipos de artefatos textuais para documentar recursos do produto, demonstrando as preferências de cada contribuidor. No entanto, os autores concluem que a falta de vinculação entre a documentação e o código-fonte pode limitar a compreensão e a capacidade de manutenção dos projetos ao longo do tempo. Assim, uma abordagem simples para alcançar uma melhor conexão entre a documentação, código e promover a colaboração seria uma seção no topo no arquivo *README* que descreva a estrutura do projeto e uma prática seria o uso de *hiperlinks* que facilite os leitores a navegar com eficiência entre a documentação dos recursos do produto e o código-fonte.

Nesse contexto, Prana *et al.* (2019) [PS06] realizaram uma abordagem de classificação automatizada para categorizar o conteúdo do *README* do *GitHub*. O classificador automatizado construído usa um método de relevância binária para classificação multi-rótulo, onde transforma cada problema de classificação multi-rótulo em um conjunto de classificações binárias. Os autores usaram como recursos heurísticos binários os padrões linguísticos, cabeçalho

de palavra única em idioma diferente do inglês, nome do repositório e texto de conteúdo não ASCII. As categorias mais presentes na classificação são: i) categoria 'O quê' com base em títulos como "Introdução" e "Sobre"ela faz uma breve introdução sobre o projeto; ii) categoria 'Como', inclui instruções sobre como usar o projeto, com conteúdo relacionado à programação (por exemplo, configuração, instalação, dependências e erros/*bugs*) e iii) categoria 'Quem' inclui informações sobre licença, detalhes de contato, e código de conduta. Com base na amostra dada pelos autores, 97% dos os arquivos contêm pelo menos uma seção descrevendo o 'O quê' do repositório e 88,5% oferecem algum conteúdo 'Como'. Além de classificar automaticamente o conteúdo, o classificador pode permitir acesso às informações não estruturadas contidas em um *README* do *GitHub* e rotular visualmente os tipos de artefatos com base nas informações contidas neles.

Ademais, a **Adoção de Inteligencia Artificial** também surgiu como uma possível tendência na documentação de software para projetos FOSS. Essa tendência envolve o aproveitamento da inteligência artificial para automatizar e melhorar vários aspectos da criação, gerenciamento e acessibilidade de documentação. Carvalho *et al.* (2014) [PS01] introduz que mineração de dados entra como uma abordagem que fornece uma ferramenta sistemática para coletar métricas sobre o conteúdo das documentações de software e avaliar a sua qualidade. As métricas de qualidade são processadas de acordo com características pré-estabelecidas pelos autores como a legibilidade, atualidade e completude das documentações. Através do Software de código aberto de Mineração de Documentação (DMOSS), é possível extrair *insights* e informações úteis em relação a documentação. Ele começa coletando o conteúdo escrito em linguagem natural, processando-o para calcular métricas e, finalmente, analisando essas métricas para tirar conclusões. Os dados são analisados a partir da estrutura de Árvore Anotada. Nesta árvore, os nós representam arquivos e diretórios, e as arestas descreve a estrutura hierárquica do pacote. A árvore é gerada automaticamente pelo DMOSS, funciona atravessando recursivamente a hierarquia de arquivos do sistema de arquivos, adicionando nós para cada arquivo e diretório. Para cada nó que tenha um formato de documentação o conteúdo de texto simples é extraído e adicionado ao nó correspondente como um atributo. A principal contribuição do DMOSS é prover uma análise documental estruturada que pode ser usada para implementar e integrar rapidamente novas métricas e análises as documentações. Assim, busca fornecer informações sobre o conteúdo encontrado nos artefatos e avalia a qualidade de acordo com a compreensão ordenada do conhecimento concedido nos artefatos.

Já Aversano *et al.* (2017) [PS04] centralizou seu estudo na avaliação da qualidade dos vários tipos de documentos através de métricas e indicadores de qualidade. Segundo os autores, a avaliação dessas métricas requer a aplicação de técnicas de Processamento de Linguagem Natural, Extração de Informação e Recuperação de Informação para fazer uma análise objetiva e não subjetiva. As métricas principais são i) Completude: esse indicador verifica se a documentação descreve todos os itens (pacotes, classes, métodos) do código-fonte; ii) Alinhamento: verifica se a documentação está atualizada de acordo com a *release* atual; iii) Dimensão: analisa se as frases do documento são muito longas ou muito curtas, com o objetivo de verificar se o texto é muito difícil de ser compreendido; iv) Estrutura: visa avaliar a estrutura em termos de número de capítulos, seções, aninhamento de subseções, extensão do documento e densidade de tabelas e figuras e v) Legibilidade: examina se as frases expressam conceitos claros e compreensíveis. Por outro lado, Ciurumelea *et al.* (2023) [PS12] utilizou modelos de Linguagem Neural para analisar elementos estruturais nos comentários de documentação e avaliar como comentários de código podem auxiliar os desenvolvedores na redação de documentações. Os modelos usados foram o Modelo LM Sequencia, Modelo LM de Contexto, Modelo LM de Contexto de Seção e LMs de contexto específicos de seção. Os modelos foram avaliados utilizando a técnica *Top-k* para atestar seu funcionamento. De acordo com as análises dos autores, a prática de comentários de documentação é amplamente utilizada, especialmente em projetos Python e Java. Assim, ter elementos estruturais não apenas melhora a legibilidade dos artefatos, mas também permite o processamento automatizado para gerar documentação ou fornecer métricas que auxilie o suporte em ambientes de desenvolvimento.

Ma *et al.* (2018) [PS05] propõe uma abordagem automatizada baseada em técnicas de *Machine Learning* para identificar os diversos tipos de artefatos de software. Foram utilizadas abordagens de árvores de decisão, máquinas de vetores de suporte e redes *bayesianas*. Usando heurística, os autores categorizaram os artefatos em dois grupos: aqueles que podem ser classificados apenas com base no nome e extensão do arquivo, por exemplo, arquivos *bat* e aqueles que requerem uma análise mais profunda para serem classificados, por exemplo, documentos de texto. Assim, os resultados do estudo empírico indicam que a abordagem automatizada baseada em técnicas de *Machine Learning* foi capaz de classificar automaticamente artefatos de software com uma precisão média de 85% e *recall* de 82% usando validação cruzada de 10 vezes no conjunto de dados de validação. Além disso, é mostrado que além do código fonte, cerca de 14,88% dos projetos de código aberto contêm outras formas de artefatos, como documentos de requisitos e

documentos de arquitetura, que são de interesse dos pesquisadores de engenharia de software. Em relação as documentações de arquitetura, Ding *et al.* (2014a) [PS02] afirmaram que esse tipo de documentação é essencial para promover a colaboração anárquica e, ao mesmo tempo, preservar o controle centralizado sobre as interfaces em projetos de código aberto bem-sucedidos. Os autores concluíram, a partir de uma pesquisa exploratória, que a probabilidade de um projeto FOSS documentar a arquitetura do software aumenta de acordo com a quantidade de desenvolvedores envolvidos no projeto, sugerindo que a documentação arquitetural parece mais útil para projetos maiores. Assim, quantidade de documentação de arquitetura depende em grande parte dos fatores contextuais de desenvolvimento, sendo o modelo arquitetural de elementos, sistema e a missão do projeto as informações arquitetônicas frequentemente documentadas.

Finalmente, tem-se estabelecido uma categoria focada em **Ferramentas e Abordagens de Apoio**. Esta categoria abrange ferramentas e abordagens projetadas para auxiliar na melhoria das práticas de documentação de software em projetos FOSS. Nesse sentido, AlOmar *et al.* (2021) [PS07] realizaram um estudo exploratório sobre como os desenvolvedores documentam suas atividades de refatoração em mensagens de *commit*. Os autores usaram a ferramenta *Refactoring Miner* para identificar as operações de refatoração que ocorrem nos projetos. O *Refactoring Miner* itera sobre o histórico de *commit* de um repositório em ordem cronológica e compara as alterações feitas nos arquivos de código-fonte para detectar refatorações. Eles descobriram que os desenvolvedores tendem a usar uma variedade de padrões textuais para documentar suas atividades de refatoração, como refatorar, mover e extrair. Classificar automaticamente um grande conjunto de *commit* contendo atividades de refatoração desencadeia motivações que vão além da necessidade básica de melhoria das documentações do sistema, pois demostram práticas direcionadores para as atividades de refatoração que incluem as seguintes categorias: Funcional, Correção de *Bug*, Atributo de Qualidade Interno, Resolução de *Code smells* e Atributo de Qualidade Externo. Os autores finalizam explanando que os colaboradores que frequentemente refatoram o código tendem a documentar menos as alterações do que os desenvolvedores que realizam a refatoração ocasionalmente.

Pasuksmit *et al.* (2022) [PS08] propuseram um modelo de predição denominado *DocWarn* para estimar a probabilidade de alterações nas documentações durante o tempo da *sprint*. Aplicando os critérios de diferença semântica do *DocWarn*, com base na avaliação qualitativa dos autores, 40% a 68% das alterações de documentação previstas estão relacionadas à modificação do escopo. Com a estimativa da probabilidade do *DocWarn*, a equipe estará

mais consciente das possíveis alterações na documentação durante o planejamento da *sprint*, permitindo que a equipe gerencie a incerteza e reduza o risco de estimativas errôneas de esforço e planejamento. Wermke *et al.* (2023) [PS11] complementa que a qualidade e a documentação disponível para um softawre FOSS parecem estar muitas vezes diretamente correlacionada com a popularidade do projeto. Além disso, projetos mais populares tem índices mais altos de horas dedicadas para a manutenção e criação da documentação.

Por fim, Bigliardi *et al.* (2014) [PS03] através de um estudo quantitativo investigam se o esforço de manutenção nas documentações aumenta à medida que sistema evolui. Os autores analisaram a correlação de classificação entre a idade do projeto (ou seja, time) e a evolução da porcentagem de *commit* nos artefatos utilizando ferramentas estatísticas, como o teste de normalidade de *Shapiro-Wilk* e o coeficiente de Gini para análise dos dados. A análise foi feita em todos os *commits* do mais antigo ao mais recente, sejam em atualizações ou remoção de arquivos. O comportamento é intrinsecamente relativo ao tipo de projeto, em alguns o esforço de manutenção aumenta com o tempo e em outros não.

5 DISCUSSÃO

Ao conduzir este Mapeamento Sistemático da Literatura (MSL), objetivamos mapear os desafios e soluções referentes ao processo de documentação em projetos de software livre e de código aberto (em inglês, *free and open-source software* ou FOSS) nos últimos dez anos. Este objetivo é justificado como relevante dado que a má documentação do software é a causa de muitos erros, desistência de manutenção no código e redução de eficiência no desenvolvimento e uso do software (DING *et al.*, 2014b). Conforme discutido por Aghajani *et al.* (2019), a documentação de software fornece aos desenvolvedores e usuários uma descrição do que um sistema de software faz, como funciona e como deve ser usado. Assim, é fundamental compreender os desafios e soluções associados à documentação em projetos FOSS dentro do seu contexto específico.

Os desafios identificados remetem a um conjunto de fatores que culminam em dificuldades lidadas pelos projetos FOSS. Detalhamos na Figura 6 os desafios das documentação retratados em nossos estudos primários, que são **Colaboração, Qualidade, Incompletude, Manutenibilidade e Categorização**. Cada desafio tem suas nuances, mas todos convergem quando se discute o papel da documentação nos projetos FOSS. Vale ressaltar que a documentação também contribui para registrar a evolução do software, estabelecendo as bases para processos subsequentes, como treinamento, uso e manutenção (FORWARD, 2002; NUNES *et al.*, 2004).

A **Colaboração** refletiu particularidades bem conhecidas enfrentadas pelo FOSS, incluindo equipes dispersas, diversos níveis de habilidade e a necessidade de atrair e reter novos colaboradores (ABERDOUR, 2007; CROWSTON *et al.*, 2007). Este emaranhado dinamismo do desenvolvimento de software pode levar a revisões contínuas devido a prioridades conflitantes e níveis variados de especialização, resultando em inconsistências de documentos (SUN *et al.*, 2023) [PS10]. Embora os padrões colaborativos favorecessem principalmente os arquivos de teste em vez da documentação em FOSS, a importância da documentação na promoção de comunidades colaborativas permaneceu subestimada (PUHLFÜRSS *et al.*, 2022) [PS09]. As questões da **qualidade** na documentação também constituíram outro desafio. Apesar de ser um recurso indispensável, a documentação muitas vezes não atende às necessidades dos desenvolvedores devido a imprecisões e ambiguidades (AVERSANO *et al.*, 2017) [PS04]. Neste sentido, a avaliação da qualidade da documentação revelou-se complexa devido à sua natureza subjetiva e envolve múltiplos fatores (legibilidade, atualidade, etc) (CARVALHO *et al.*, 2014; PLÖSCH *et al.*, 2014).

O desafio da **Incompletude** acentuou a deficiência de informações essenciais na documentação que são necessárias para as tarefas dos profissionais (ZHI *et al.*, 2015). Na verdade, fóruns e mídias sociais tornaram-se fontes alternativas de informação para desenvolvedores, mas a escassez de arquitetura abrangente e documentação de refatoração persistiu como uma questão em aberto em projetos FOSS (ALOMAR *et al.*, 2021; DING *et al.*, 2014a). Além disso, a falta de padrões sistemáticos para arquivos *README* evidencia a escassez de documentação relevante (PRANA *et al.*, 2019) [PS06]. A perspectiva da **Manutenção** também destacou a necessidade de proporcionar facilidade para fazer alterações ou correções em documentos (AGHAJANI *et al.*, 2020). Enquanto alguns projetos exibiam correlações positivas entre esforço de manutenção e evolução da documentação, outros não (BIGLIARDI *et al.*, 2014) [PS03]. A **Categorização** dos tipos de documentação também se mostrou relevante ao contribuir para a organização e classificação de diversos documentos, como manuais do usuário, especificações técnicas e documentação de API, refletindo a natureza multifacetada da documentação em projetos de software de código aberto (AGHAJANI *et al.*, 2019; MA *et al.*, 2018). Além disso, o nível de suporte de documentação parecia correlacionar-se com o tamanho da equipe e os níveis de experiência, destacando as dependências contextuais (WERMKE *et al.*, 2023) [PS11].

As soluções que surgiram de nossos estudos primários (ver Figura 7) incluem o **Uso Estratégico do README, a Adoção de Inteligência Artificial e Ferramentas e Abordagens de Apoio**. Essas soluções demonstram os esforços contínuos para aprimorar práticas de documentação e apoiar colaboração eficiente e manutenção em iniciativas de software livre e de código aberto (FOSS). Em particular, o **Uso Estratégico do README** em projetos FOSS surgiu como um artefato a ser observado. Esses arquivos geralmente contêm informações sobre o projeto, como instruções de instalação, diretrizes de uso e recursos principais (KOSKELA *et al.*, 2018). Ao utilizar estrategicamente os arquivos README, os projetos FOSS podem aprimorar a acessibilidade, melhorar a integração de usuários e facilitar a colaboração eficaz entre os membros da equipe (LIU *et al.*, 2022).

Uma outra perspectiva de solução ganhando destaque surge da **Adoção de Inteligência Artificial**. Essa tendência envolve aproveitar a inteligência artificial para automatizar e aprimorar a criação, gestão e acessibilidade da documentação. Ciurumelea *et al.* (2023) [PS12], por exemplo, exploraram modelos de linguagem neural para analisar elementos estruturais em comentários de documentação, avaliando como os comentários de código podem ajudar os desenvolvedores a escrever documentação.

Por fim, **Ferramentas e Abordagens de Apoio** foram identificadas como instrumentais para melhorar as práticas de documentação de software em projetos FOSS. AlOmar *et al.* (2021) [PS07] realizaram um estudo exploratório sobre a documentação das atividades de refatoração pelos desenvolvedores em mensagens de *commit* com o apoio da ferramenta *Refactoring Miner*. Além disso, Pasuksmit *et al.* (2022) [PS08] propuseram o DocWarn, um modelo de previsão que estima a probabilidade de mudanças na documentação durante o tempo de *sprint*, auxiliando na estimativa de esforço e na redução de riscos.

Assim, analisamos a relação entre desafios e soluções identificados. A utilização estratégica de arquivos *README* exemplifica uma abordagem essencial para agilizar esforços colaborativos. Especificamente, os arquivos *README* facilitam a integração mais suave de novos participantes e servem como resumos concisos de categorias de documentação. Além disso, a integração de inteligência artificial introduz a geração automatizada de documentação por meio da análise de código, melhorando assim a qualidade da documentação e garantindo sua relevância atual, abordando assim a potencial incompletude. Além disso, a implementação de Ferramentas e Abordagens de Apoio aprimora a manutenção da documentação por meio de vários mecanismos, incluindo automação, integração com sistemas de controle de versão, recursos de edição colaborativa, padronização de modelos e promoção de melhores práticas. Notavelmente, ferramentas automatizadas como *Sphinx* ou *Javadoc* exemplificam essa abordagem atualizando automaticamente a documentação em resposta a modificações de código, mitigando assim a intervenção manual.

Como podemos ver, as descobertas deste trabalho têm implicações além do contexto acadêmico e podem ser úteis na indústria de software. Compreender os desafios e soluções pode ajudar os interessados no FOSS a tomar decisões informadas sobre a priorização dos esforços de documentação, incentivando a colaboração e utilizando abordagens eficientes. Com sorte, esses resultados podem contribuir para promover uma reflexão concreta e aumentar a conscientização sobre o processo de documentação em projetos FOSS e, consequentemente, liderar esforços para melhorar a qualidade, acessibilidade e sustentabilidade das práticas de documentação de software, em consonância com as demandas e complexidades em evolução do FOSS.

6 AMEAÇAS À VALIDADE

Para identificar e categorizar as ameaças à validade e as ações de mitigação correspondentes, utilizamos o *checklist* proposto por Ampatzoglou *et al.* (2019) para identificar ameaças à validade em estudos secundários na Engenharia de Software.

Viés de inclusão/exclusão dos estudos: Este trabalho reconhece que pode haver artigos relevantes sobre o uso de documentação de software em projetos FOSS que não estão em nosso quadro de amostragem, por exemplo, artigos que não contemplam o período de publicações entre 2013 e 2023 e artigos que não use os termos que incluímos na *string* de busca da pesquisa. No entanto, medidas foram tomadas para mitigar essa ameaça. Inicialmente, justificamos explicitamente este intervalo de tempo como estratégico, pois nos permite focar em perspectivas recentes. Além disso, esta decisão é claramente declarada na pergunta de pesquisa. O período entre 2013 e 2023 foi especificado nos filtros de cada banco de dados usado na busca, considerando o foco deste MSL. Mesmo com essa restrição, vale ressaltar o número considerável de artigos (1271) obtidos após a execução da busca. Também elaboramos uma *string* de busca combinando vários termos e sinônimos relacionados a FOSS e documentação de software. Esta questão incluiu diferentes variações como *open source* e *open-source*, bem como siglas como OSS e FOSS. Termos relacionados à documentação ou documentos também foram incluídos. Além disso, utilizamos uma técnica de *snowballing backward* para identificar estudos potencialmente perdidos com base nas listas de referência dos artigos obtidos.

Viés do pesquisador e repetibilidade: O processo de busca explorou rigorosamente nossa *string* de busca em diferentes bancos de dados bem conhecidos, aumentando assim a repetibilidade e transparência da pesquisa. O primeiro autor conduziu a busca inicial em cada banco de dados e removeu os artigos duplicados. Na aplicação dos Critérios de Inclusão (CI) e Critérios de Exclusão (CE), a lista completa de artigos foi dividida em dois grupos, e outros dois pesquisadores convidados aplicaram os CIs e CEs, enquanto o autor principal realizou uma verificação dupla em todos os artigos (de ambos os grupos). Os pesquisadores se encontraram virtualmente para resolver conflitos de decisão e alcançaram um consenso preliminar de seleção. Dois pesquisadores mais experientes foram convidados a discutir a decisão final se os conflitos persistissem. Esse processo contribuiu para aumentar a precisão dos dados.

Robustez da classificação: Nossa análise qualitativa dos desafios e soluções pode representar uma ameaça potencial devido à subjetividade inerente. Embora a análise tenha sido baseada em descobertas de artigos selecionados, o processo de categorização é influenciado

pelas interpretações dos autores, o que é natural e valioso em pesquisa qualitativa (DYBÅ *et al.*, 2011). No entanto, esse processo de codificação aberta passou por discussões entre o autor principal e outros dois pesquisadores com extensa experiência em pesquisa em Engenharia de Software para fortalecer a compreensão, rigor e relevância.

7 CONSIDERAÇÕES FINAIS

Neste estudo investigamos os desafios e soluções relacionados as documentações de software em projetos de software livre e de código aberto (em inglês, *free and open-source software* ou FOSS). Através de um Estudo de Mapeamento Sistemático (MSL) focado nos artigos publicados entre 2013 e 2023, uma *string* de busca foi explorada sistematicamente para selecionar os estudos primários, complementados por *snowballing backward*. Nossa estratégia de busca resultou em um total de 12 estudos primários. As descobertas revelaram cinco grandes desafios (Colaboração, Qualidade, Incompletude, Manutenibilidade e Categorização) e três perspectivas de solução abrangentes (Uso Estratégico de README, Adoção de Inteligência Artificial e Ferramentas e Abordagens de Apoio) pertinentes à documentação em FOSS. Essas descobertas nos ajudaram a responder nossa pergunta de pesquisa: *Nos últimos dez anos, quais são os desafios e soluções relacionados à documentação de software em FOSS?*

Este estudo oferece contribuições inovadoras tanto para a academia quanto para a prática. Do ponto de vista acadêmico, esta pesquisa oferece um MSL revelando um conjunto de desafios e soluções relacionadas à documentação de software, um tópico ainda pouco explorado no contexto de software de código aberto. Tal abordagem fornecerá uma visão abrangente da área, contribuindo para a consolidação do conhecimento e a identificação de lacunas em pesquisas anteriores. Além disso, os resultados obtidos reforçam possibilidades de estudo, fornecendo uma base valiosa para futuras pesquisas nesse campo de interseção. Do ponto de vista prático, este trabalho promove uma reflexão sobre o uso das documentações em projetos FOSS, destacando desafios e soluções que podem aprimorar o processo de qualidade das documentações e eficiência dos projetos FOSS.

Em termos de trabalhos futuros, poderiam investigar a aplicabilidade dos desafios e soluções identificados em nosso MSL em projetos no *GitHub*. Esta investigação poderia fornecer descobertas valiosas sobre como esses desafios e soluções se manifestam e são abordados dentro de uma plataforma que hospeda diversos projetos de software. Além disso, pesquisas futuras poderiam eventualmente explorar os contrastes e semelhanças nas práticas de documentação de software entre projetos de código fechado e FOSS.

REFERÊNCIAS

ABERDOUR, M. Achieving quality in open-source software. **IEEE software**, IEEE, v. 24, n. 1, p. 58–64, 2007.

AGHAJANI, E.; NAGY, C.; LINARES-VÁSQUEZ, M.; MORENO, L.; BAVOTA, G.; LANZA, M.; SHEPHERD, D. C. Software documentation: the practitioners' perspective. In: **Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering**. [S.l.: s.n.], 2020. p. 590–601.

AGHAJANI, E.; NAGY, C.; VEGA-MÁRQUEZ, O. L.; LINARES-VÁSQUEZ, M.; MORENO, L.; BAVOTA, G.; LANZA, M. Software documentation issues unveiled. In: IEEE. **2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)**. [S.l.], 2019. p. 1199–1210.

ALOMAR, E. A.; PERUMA, A.; MKAOUER, M. W.; NEWMAN, C. D.; OUNI, A. Behind the scenes: On the relationship between developer experience and refactoring. **arXiv e-prints**, p. arXiv–2109, 2021.

AMBLER, S. W. Agile documentation. 2001-2004. **The Official Agile Modeling (AM) Site**, 2001.

AMPATZOGLOU, A.; BIBI, S.; AVGERIOU, P.; VERBEEK, M.; CHATZIGEORGIOU, A. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. **Information and Software Technology**, Elsevier, v. 106, p. 201–230, 2019.

ANGEREN, J. V.; KABBEDIJK, J.; JANSEN, S.; POPP, K. M. A survey of associate models used within large software ecosystems. In: **IWSECO@ ICSOB**. [S.l.: s.n.], 2011. p. 27–39.

AVELINO, G.; CONSTANTINOU, E.; VALENTE, M. T.; SEREBRENIK, A. On the abandonment and survival of open source projects: An empirical investigation. In: IEEE. **2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)**. [S.l.], 2019. p. 1–12.

AVERSANO, L.; GUARDABASCIO, D.; TORTORELLA, M. Evaluating the quality of the documentation of open source software. In: SCITEPRESS. **International Conference on Evaluation of Novel Approaches to Software Engineering**. [S.l.], 2017. v. 2, p. 308–313.

BIGLIARDI, L.; LANZA, M.; BACCHELLI, A.; D'AMBROS, M.; MOCCI, A. Quantitatively exploring non-code software artifacts. In: IEEE. **2014 14th International Conference on Quality Software**. [S.l.], 2014. p. 286–295.

BRIAND, L. C. Software documentation: how much is enough? In: IEEE. **Seventh European Conference onSoftware Maintenance and Reengineering, 2003. Proceedings.** [S.l.], 2003. p. 13–15.

CARVALHO, N. R.; SIMOES, A.; ALMEIDA, J. J. Dmoss: Open source software documentation assessment. **Computer Science and Information Systems**, v. 11, n. 4, p. 1197–1207, 2014.

CHAPIN. Trends in preserving and enhancing the value of software. In: IEEE. **Proceedings 2000 International Conference on Software Maintenance**. [S.l.], 2000. p. 6–8.

CIOCH, F. A.; PALAZZOLO, M.; LOHRER, S. A documentation suite for maintenance programmers. In: IEEE COMPUTER SOCIETY. **1996 Proceedings of International Conference on Software Maintenance**. [S.l.], 1996. p. 286–286.

CIURUMELEA, A.; ALEXANDRU, C. V.; GALL, H. C.; PROKSCH, S. Completing function documentation comments using structural information. **Empirical Software Engineering**, Springer, v. 28, n. 4, p. 86, 2023.

COELHO, H. S. Documentação de software: uma necessidade. **Texto Livre: linguagem e tecnologia**, Universidade Federal de Minas Gerais, v. 2, n. 1, p. 17–21, 2009.

CROWSTON, K.; LI, Q.; WEI, K.; ESERYEL, U. Y.; HOWISON, J. Self-organization of teams for free/libre open source software development. **Information and software technology**, Elsevier, v. 49, n. 6, p. 564–575, 2007.

DAI, J. X.; BOUJUT, J.-F.; POURROY, F.; MARIN, P. Issues and challenges of knowledge management in online open source hardware communities. **Design Science**, Cambridge University Press, v. 6, p. e24, 2020.

DING, W.; LIANG, P.; TANG, A.; VLIET, H. V.; SHAHIN, M. How do open source communities document software architecture: An exploratory survey. In: IEEE. **2014 19th International conference on engineering of complex computer systems**. [S.l.], 2014. p. 136–145.

DING, W.; LIANG, P.; TANG, A.; van Vliet, H. Knowledge-based approaches in software documentation: A systematic literature review. **Information and Software Technology**, v. 56, n. 6, p. 545–567, 2014. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584914000196>.

DING, W.; LIANG, P.; TANG, A.; VLIET, H. V. Knowledge-based approaches in software documentation: A systematic literature review. **Information and Software Technology**, Elsevier, v. 56, n. 6, p. 545–567, 2014.

DYBÅ, T.; PRIKLADNICKI, R.; RÖNKKÖ, K.; SEAMAN, C.; SILLITO, J. Qualitative research in software engineering. **Empirical Software Engineering**, Springer, v. 16, p. 425–429, 2011.

FORWARD, A. **Software documentation: Building and maintaining artefacts of communication.** [S.l.]: University of Ottawa (Canada), 2002.

FRANCO-BEDOYA, O.; AMELLER, D.; COSTAL, D.; FRANCH, X. Open source software ecosystems: A systematic mapping. **Information and software technology**, Elsevier, v. 91, p. 160–185, 2017.

GAROUSI, G. **A Hybrid Methodology for Analyzing Software Documentation Quality and Usage**. Dissertação (Mestrado) — Graduate Studies, 2012.

GAROUSI, G.; GAROUSI, V.; MOUSSAVI, M.; RUHE, G.; SMITH, B. Evaluating usage and quality of technical software documentation: an empirical study. In: **Proceedings of the 17th international conference on evaluation and assessment in software engineering**. [S.l.: s.n.], 2013. p. 24–35.

HABIB, B.; ROMLI, R. A systematic mapping study on issues and importance of documentation in agile. In: IEEE. **2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS)**. [S.l.], 2021. p. 198–202.

HUMES, L. L. Communities of practice for open source software. In: **Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives**. [S.l.]: IGI Global, 2007. p. 610–623.

ISKOUJINA, Z.; ROBERTS, J. Knowledge sharing in open source software communities: motivations and management. **Journal of Knowledge Management**, Emerald Group Publishing Limited, 2015.

ISLAM, M. A.; HASAN, R.; EISTY, N. U. Documentation practices in agile software development: A systematic literature review. In: IEEE. **2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)**. [S.l.], 2023. p. 266–273.

IZQUIERDO, J. L. C.; CABOT, J. On the analysis of non-coding roles in open source development: An empirical study of npm package projects. **Empirical Software Engineering**, Springer, v. 27, n. 1, p. 18, 2022.

JALOTE, P. **An integrated approach to software engineering**. [S.l.]: Springer Science & Business Media, 2012.

KAUR, R.; CHAHAL, K. K.; SAINI, M. Understanding community participation and engagement in open source software projects: A systematic mapping study. **journal of king saud university-computer and information sciences**, Elsevier, v. 34, n. 7, p. 4607–4625, 2022.

KITCHENHAM, B.; CHARTERS, S. *et al.* Guidelines for performing systematic literature reviews in software engineering version 2.3. **Engineering**, v. 45, n. 4ve, p. 1051, 2007.

KLUG, D.; BOGART, C.; HERBSLEB, J. D. "they can only ever guide"how an open source software community uses roadmaps to coordinate effort. **Proceedings of the ACM on Human-Computer Interaction**, ACM New York, NY, USA, v. 5, n. CSCW1, p. 1–28, 2021.

KOSKELA, M.; SIMOLA, I.; STEFANIDIS, K. Open source software recommendations using github. In: SPRINGER. **Digital Libraries for Open Knowledge: 22nd International Conference on Theory and Practice of Digital Libraries, TPDL 2018, Porto, Portugal, September 10–13, 2018, Proceedings 22**. [S.l.], 2018. p. 279–285.

KROGH, G. V.; HIPPEL, E. V. The promise of research on open source software. **Management science**, INFORMS, v. 52, n. 7, p. 975–983, 2006.

LAKULU, M.; ABDULLAH, R.; SELAMAT, M. H.; IBRAHIM, H.; NOR, M. Z. M. A framework of collaborative knowledge management system in open source software development environment. **Computer and Information Science**, Citeseer, v. 3, n. 1, p. 81, 2010.

LIU, Y.; NOEI, E.; LYONS, K. How readme files are structured in open source java projects. **Information and Software Technology**, Elsevier, v. 148, p. 106924, 2022.

MA, Y.; FAKHOURY, S.; CHRISTENSEN, M.; ARNAOUDOVA, V.; ZOGAAN, W.; MIRAKHORLI, M. Automatic classification of software artifacts in open-source applications. In: **Proceedings of the 15th International Conference on Mining Software Repositories**. [S.l.: s.n.], 2018. p. 414–425.

MATTE, A. C. F. Uma definição informal de documentação: análise semiótica. **Texto Livre: linguagem e tecnologia**, Universidade Federal de Minas Gerais, v. 1, n. 2, p. 45–59, 2008.

MICHELAZZO, P. **Documentação de software, 2006**. 2008.

MICHLMAYR, M.; HUNT, F.; PROBERT, D. Quality practices and problems in free software projects. In: **Proceedings of the first international conference on open source systems**. [S.l.: s.n.], 2005. p. 24–28.

MICHLMAYR, M.; HUNT, F.; PROBERT, D. Release management in free software projects: Practices and problems. In: SPRINGER. **Open Source Development, Adoption and Innovation: IFIP Working Group 2.13 on Open Source Software, June 11–14, 2007, Limerick, Ireland 3**. [S.l.], 2007. p. 295–300.

NUNES, V. B.; SOARES, A. O.; FALBO, R. A. Apoio à documentação em um ambiente de desenvolvimento de software. In: **Memorias de VII Workshop Iberoamericano de Ingeniería de Requisitos y Desarrollo de Ambientes de Software-IDEAS**. [S.l.: s.n.], 2004. p. 50–55.

PARNAS, D. L. Precise documentation: The key to better software. In: **The future of software engineering**. [S.l.]: Springer, 2010. p. 125–148.

PASUKSMIT, J.; THONGTANUNAM, P.; KARUNASEKERA, S. Towards reliable agile iterative planning via predicting documentation changes of work items. In: **Proceedings of the 19th International Conference on Mining Software Repositories**. [S.l.: s.n.], 2022. p. 35–47.

PERENS, B. *et al.* The open source definition. **Open sources: voices from the open source revolution**, Sebastopol, CA: O'Reilly, v. 1, p. 171–188, 1999.

PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. Systematic mapping studies in software engineering. In: **12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12**. [S.l.: s.n.], 2008. p. 1–10.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. **Information and software technology**, Elsevier, v. 64, p. 1–18, 2015.

PINHO, G. **Supporting repository**. 2024. Disponível em: <https://zenodo.org/records/11053588>.

PINTO, V. H. M. O papel da documentação no desenvolvimento de software open source: Uma análise e um estudo de caso. 2021.

PLÖSCH, R.; DAUTOVIC, A.; SAFT, M. The value of software documentation quality. In: IEEE. **2014 14th International Conference on Quality Software**. [S.l.], 2014. p. 333–342.

PRANA, G. A. A.; TREUDE, C.; THUNG, F.; ATAPATTU, T.; LO, D. Categorizing the content of github readme files. **Empirical Software Engineering**, Springer, v. 24, p. 1296–1327, 2019.

PUHLFÜRSS, T.; MONTGOMERY, L.; MAALEJ, W. An exploratory study of documentation strategies for product features in popular github projects. In: IEEE. **2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)**. [S.l.], 2022. p. 379–383.

SALDAÑA, J. The coding manual for qualitative researchers. SAGE publications Ltd, 2021.

SCHWEIGER, D. M.; SANDBERG, W. R.; RAGAN, J. W. Group approaches for improving strategic decision making: A comparative analysis of dialectical inquiry, devil's advocacy, and consensus. **Academy of management Journal**, Academy of Management Briarcliff Manor, NY 10510, v. 29, n. 1, p. 51–71, 1986.

SILVA, A. S. d. F. **Documentação de software: uma análise comparativa entre documentação tradicional e living documentation**. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, 2020.

SOUZA, S. C. B. de; ANQUETIL, N.; OLIVEIRA, K. M. de. A study of the documentation essential to software maintenance. In: **Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information**. [S.l.: s.n.], 2005. p. 68–75.

STAMELOS, I.; ANGELIS, L.; OIKONOMOU, A.; BLERIS, G. L. Code quality analysis in open source software development. **Information systems journal**, Wiley Online Library, v. 12, n. 1, p. 43–60, 2002.

STEINMACHER, I.; CHAVES, A. P.; CONTE, T. U.; GEROSA, M. A. Preliminary empirical identification of barriers faced by newcomers to open source software projects. In: IEEE. **2014 Brazilian Symposium on Software Engineering**. [S.l.], 2014. p. 51–60.

SUN, W.; IWUCHUKWU, S.; BANGASH, A. A.; HINDLE, A. An empirical study to investigate collaboration among developers in open source software (oss). In: IEEE. **2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)**. [S.l.], 2023. p. 352–356.

THEUNISSEN, T.; HEESCH, U. van; AVGERIOU, P. A mapping study on documentation in continuous software development. **Information and software technology**, Elsevier, v. 142, p. 106733, 2022.

TRINKENREICH, B. Please don't go—a comprehensive approach to increase women's participation in open source software. In: IEEE. **2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)**. [S.l.], 2021. p. 293–298.

WERMKE, D.; KLEMMER, J. H.; WÖHLER, N.; SCHMÜSER, J.; RAMULU, H. S.; ACAR, Y.; FAHL, S. "always contribute back": A qualitative study on security challenges of the open source supply chain. In: IEEE. **2023 IEEE Symposium on Security and Privacy (SP)**. [S.l.], 2023. p. 1545–1560.

WEST, J.; BOGERS, M. Leveraging external sources of innovation: A review of research on open innovation. **Journal of product innovation management**, Wiley Online Library, v. 31, n. 4, p. 814–831, 2014.

WINTERS, T.; MANSHRECK, T.; WRIGHT, H. **Software engineering at google: Lessons learned from programming over time**. [S.l.]: O'Reilly Media, 2020.

WU, M.-W.; LIN, Y.-D. Open source software development: An overview. **Computer**, IEEE, v. 34, n. 6, p. 33–38, 2001.

ZHI, J.; GAROUSI-YUSIFOĞLU, V.; SUN, B.; GAROUSI, G.; SHAHNEWAZ, S.; RUHE, G. Cost, benefits and quality of software development documentation: A systematic mapping. **Journal of Systems and Software**, Elsevier, v. 99, p. 175–198, 2015.