# Capítulo 3

# Princípios e Práticas para Sustentabilidade do Software de Pesquisa

Christina von Flach, Joenio M. Costa e Daniela Feitosa

***Abstract***

*The growing use of Research Software, that is, software developed in the context of scientific research, has awakened in the scientific community a concern with its sustainability and influence on the ability to reproduce scientific studies by independent researchers. Software sustainability relates to the long-term consequences of designing, building and delivering a software project and concerns the software's ability to last and continue to be supported over time. However, researchers are not familiar with good practices for developing sustainable software. This course addresses the sustainability of research software from a technical perspective. We present best practices in software development that can be useful to support researchers from different areas in developing sustainable research software. We motivate and illustrate the use of practices, their benefits and related challenges using the results of a study conducted with a research group in the field of Physics.*

***Resumo***

*O uso crescente de Software de Pesquisa, isto é, software desenvolvido no contexto de uma pesquisa científica, tem despertado na comunidade científica uma preocupação com sua sustentabilidade e influência na capacidade de reprodução de estudos científicos por pesquisadores independentes. O conceito de Sustentabilidade de Software está relacionado às consequências de longo prazo de projetar, construir e entregar um projeto de software e diz respeito à capacidade do software de perdurar e ter suporte ao longo do tempo. Entretanto, pesquisadores não estão familiarizados com boas práticas de desenvolvimento de software sustentável. Este curso aborda a sustentabilidade do Software de Pesquisa sob uma perspectiva técnica. Apresentamos boas práticas de desenvolvimento de software que podem ser úteis para apoiar pesquisadores de diferentes áreas no desenvolvimento de software de pesquisa sustentável. Nós motivamos e ilustramos o uso das práticas, seus benefícios e desafios relacionados por meio dos resultados de um estudo conduzido com um grupo de pesquisa da área de Física.*

## 3.1. Introdução

Nas últimas décadas, produtos de *software* têm assumido um papel fundamental no âmbito da pesquisa científica. Na Ciência Aberta [UNESCO 2021], ao lado dos dados de pesquisa, o software desempenha um papel central, seja como parte do método científico ou como um de seus resultados [Bezjak et al. 2018].

Coletar e analisar dados, construir e testar modelos tornaram-se atividades complexas em quase todas as áreas de pesquisa, das ciências exatas às ciências humanas. [Hettrick et al. 2014] relataram que 92% dos cientistas do Reino Unido usavam software em suas pesquisas, 69% declararam que a pesquisa não poderia ser realizada sem software, e 56% desenvolviam seu próprio software, sendo que 21% destes não possuíam treinamento em desenvolvimento de software. A capacidade de lidar com tal complexidade depende de software especializado, chamado de *software científico* [Hannay et al. 2009], *software acadêmico* [Howison and Herbsleb 2011] ou *software de pesquisa*.

O termo *Software de Pesquisa*[1] foi criado para designar, de modo abrangente, o software utilizado durante a pesquisa científica, incluindo o software de terceiros usado para coleta, processamento e análise de dados [Allen et al. 2017]. Recentemente, o uso do termo foi restrito para designar apenas *o software desenvolvido especificamente no contexto da pesquisa* [Nieuwpoort and Katz 2023], com recomendação para que, ao lado dos conjuntos de dados da pesquisa, o código-fonte ou o ambiente de execução do *software de pesquisa* também sejam disponibilizados. Em tal contexto, há uma preocupação legítima por parte da comunidade científica com a qualidade do *software de pesquisa* , em especial, a sua sustentabilidade [Carver et al. 2022]. A falta de sustentabilidade do *software de pesquisa* pode ferir a reprodutibilidade na pesquisa, ou a capacidade de reprodução de estudos científicos por pesquisadores independentes, podendo ocasionar graves erros em conclusões centrais da Ciência [Merali 2010].

O conceito de *sustentabilidade de software* está relacionado às consequências de longo prazo de projetar, construir e entregar um projeto de software [Venters et al. 2014, Venters et al. 2021]. Para o *software de pesquisa*, sustentabilidade diz respeito à capacidade do software de perdurar e de continuar sendo suportado ao longo do tempo, o que implica em qualidades de longevidade e manutenibilidade. O *software de pesquisa* sustentável deve permanecer utilizável por um longo período de tempo e retornar resultados consistentes (mesmo diante de software e hardware em evolução), com benefícios para a comunidade de pesquisa. O *software de pesquisa* sustentável precisa ser atualizado, adaptado para novos ambientes e plataformas e testado.

[Carver et al. 2022] realizaram uma pesquisa com 1.149 pesquisadores para identificar os desafios relacionados a sustentabilidade enfrentados no desenvolvimento e uso de software de pesquisa. Dentre os resultados, a pesquisa identificou que (i) o *software de pesquisa* é desenvolvido e gerenciado sem considerar sua sustentabilidade no longo prazo e, (ii) há necessidade de treinamento para os desenvolvedores de *software de pesquisa* sobre o ciclo de vida do software e ferramentas para seu desenvolvimento e manutenção. De fato, grande parte dos cientistas que escrevem *software de pesquisa* ainda carecem de uma

[1]Tradução nossa para *Research Software*. O termo *Software de Pesquisa* será utilizado ao longo do texto, sempre em itálico.

formação em boas práticas de engenharia de software voltadas para o desenvolvimento de software sustentável, por exemplo, uso de sistemas de controle de versão, documentação adequada e testes automatizados. Neste cenário, a sustentabilidade do *software de pesquisa* pode ficar comprometida, levar a erros em conclusões científicas e à falta de reprodutibilidade na pesquisa.

Além das preocupações com sustentabilidade, e considerando que software é um artefato de pesquisa digital, o princípio da reprodutibilidade requer que, assim como os dados de pesquisa [Wilkinson et al. 2016], o *software de pesquisa* seja *FAIR*, isto é, facilmente localizável, acessível, interoperável e reutilizável [Chue Hong et al. 2022]. Finalmente, é importante considerar que a escolha de um software ou de uma versão do mesmo software pode ter influência nos resultados da pesquisa: as versões do software podem ter funcionalidades parecidas mas com diferenças não documentadas que podem levar as resultados diferentes.

Este capítulo apresenta uma introdução ao *software de pesquisa*, contextualizado no universo da Ciência Aberta, e caracterizado com base em princípios científicos e práticas da engenharia de software, buscando destacar o seu papel na promoção da pesquisa sustentável e reprodutível.

As definições apresentadas sobre Ciência Aberta têm como base a Recomendação da UNESCO sobre Ciência Aberta [UNESCO 2021] e o Manual de Treinamento em Ciência Aberta [Bezjak et al. 2018], disponibilizado sob a licença Creative Commons CC0 1.0 Universal (CC0 1.0) Dedicação ao Domínio Público[2]. As práticas para o desenvolvimento de *software de pesquisa* sustentável apresentadas têm como base parte do material público disponibilizado por *Software Carpentry* [Munk et al. 2019, Nenadic et al. 2022], *Library Carpentry* [Munk et al. 2019], *Netherlands eScience Center* [Drost et al. 2020] e *The Turing Way* [Community 2022]. Finalmente, os exemplos e o material de um estudo realizado com um grupo de pesquisa na área de Física foram criados pelos autores especialmente para este curso e estão disponíveis sob a licença licença Creative Commons CC0 1.0 Universal (CC0 1.0) Dedicação ao Domínio Público.

## 3.2. O que é Ciência Aberta?

O *software de pesquisa* desempenha um papel central na Ciência Aberta, seja como parte do método científico ou como um de seus resultados [Bezjak et al. 2018]. Nesta seção, apresentamos uma definição para Ciência Aberta e os conceitos mais relevantes para contextualizar o *software de pesquisa*, dentre eles, *Dados Abertos*, *Acesso Aberto*, *Revisão por Pares Aberta*, *Pesquisa Reprodutível Aberta* e *Código Aberto*.

### 3.2.1. Definição

A Recomendação da UNESCO sobre Ciência Aberta [UNESCO 2021] a define como "um construto inclusivo que combina vários movimentos e práticas que têm o objetivo de disponibilizar abertamente conhecimento científico multilíngue, torná-lo acessível e reutilizável para todos, aumentar as colaborações científicas e o compartilhamento de informações para o benefício da ciência e da sociedade, e abrir os processos de criação,

[2] `https://creativecommons.org/publicdomain/zero/1.0/`

avaliação e comunicação do conhecimento científico a atores da sociedade, além da comunidade científica tradicional".

Na definição da UNESCO, a Ciência Aberta abrange todas as disciplinas científicas e todos os aspectos das práticas acadêmicas, incluindo ciências básicas e aplicadas, ciências naturais, sociais e humanas, e está estruturada sobre quatro pilares (Figura 3.1): conhecimento científico aberto, infraestrutura científica aberta, envolvimento aberto dos atores sociais e diálogo aberto com outros sistemas de conhecimento.


**Figura 3.1. Definição de Ciência Aberta [UNESCO 2021].**

No contexto deste capítulo, que trata especificamente sobre princípios e práticas do *software de pesquisa*, colocamos ênfase em conceitos do *conhecimento científico aberto*, que versa sobre o acesso aberto a publicações científicas, dados de pesquisa, software de pesquisa, que estejam disponíveis em domínio público ou sob direitos autorais e licenciados sob uma licença aberta. Conceitos relacionados aos demais pilares estão descritos na Recomendação da UNESCO sobre Ciência Aberta [UNESCO 2021].

Consideramos que o termo *Ciência Aberta* designa a *prática da Ciência* de forma que outros possam *colaborar e contribuir*, onde publicações, dados, software e outros artefatos de pesquisa estão *disponíveis online e gratuitamente*, com base em termos que permitem a reutilização, redistribuição e *reprodução* da pesquisa, seus dados e métodos subjacentes.

### 3.2.2. Dados Abertos

Dados abertos estão acessíveis e disponíveis online, gratuitamente e podem ser usados, reutilizados e distribuídos desde que a fonte de dados seja atribuída. Pesquisadores podem

depositar os dados de sua pesquisa em um *repositório de dados aberto*, para que outros possam encontrar, utilizar e construir sobre o seu trabalho [Bezjak et al. 2018]. *Metadados* ou informações sobre os dados, tais como, criador, palavras-chave, unidades, são essenciais para facilitar a descoberta de dados e *identificadores permanentes* (PIDs) são necessários para rastrear os dados.

Um *identificador de objeto digital* (DOI) é um identificador exclusivo atribuído a um conjunto de dados para garantir que os dados não sejam perdidos ou identificados incorretamente. O DOI facilita a citação e o rastreamento do impacto dos conjuntos de dados, assim como os artigos de periódicos citados. A *citação de dados* deve ser considerada tão importante quanto a citação de artigos de periódicos e outros documentos científicos. Em geral, uma citação de dados deve incluir nome do autor/criador, data de publicação e título do conjunto de dados, editora/organizador e DOI.

Agências de fomento à pesquisa passaram a exigir que dados coletados ou criados em projetos financiados sejam compartilhados com a comunidade em geral. No Brasil, a Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) lançou as bases para a abertura de dados com a exigência, a partir do final de 2017, de Planos de Gestão de Dados quando da submissão de projetos[3].

Em geral, a organização de dados de pesquisa abertos é temática. O World Ocean Database (WOD)[4] é a maior coleção do mundo de dados abertos de perfis oceânicos uniformemente formatados, com controle de qualidade e disponíveis ao público. Os conjuntos de dados (*datasets*) submetidos recebem um número de acesso (*Accession Number*). Usuários podem utilizar o número de acesso para consultar o Geoportal do *National Centers for Environmental Information (NCEI)* e obter uma cópia exata dos dados originais e seus metadados.

### 3.2.3. Acesso Aberto

Uma das formas mais comuns de divulgar os resultados de uma pesquisa científica é escrever um artigo e publicá-lo em periódicos, anais de conferências ou livros. Em geral, tais publicações são disponibilizadas ao público mediante pagamento, através de uma assinatura institucional ou de forma individual. Entretanto, nos últimos anos, as taxas praticadas pelas editoras têm sido altíssimas, inviabilizando o amplo acesso ao conhecimento científico e criando desigualdades regionais.

O Movimento de Acesso Aberto[5] surgiu com o propósito de tornar todos os resultados de pesquisa disponíveis para o público sem quaisquer restrições. As duas estratégias mais conhecidas para publicação de artigos científicos promovem modelos alternativos, visando a ampla divulgação e redução de custos.

#### Publicação em repositórios institucionais ou temáticos

Esta estratégia recomenda o auto-arquivamento por pesquisadores, que depositam e divulgam os seus artigos em repositórios abertos – institucionais ou temáticos.

---

[3] `https://fapesp.br/gestaodedados`

[4] `https://www.ncei.noaa.gov/products/world-ocean-database`

[5] `https://en.unesco.org/open-access/open-access-movement`

No Brasil, várias universidades oferecem repositórios institucionais para que pesquisadores depositem a sua produção acadêmica. Por exemplo, o Repositório Institucional da Universidade Federal da Bahia (RI-UFBA)[6] é um serviço de informação científica que utiliza o DSpace[7], um pacote de software livre e de código aberto, para gerenciar e a disseminar a produção acadêmica da UFBA em consonância com as recomendações da Ciência Aberta.

Nos últimos anos, pesquisadores têm depositado uma versão de seus artigos pré-impressão (*preprints*) em repositórios abertos temáticos, por exemplo, o arXiv[8]. O *Computing Research Repository (CoRR) - arXiv*[9] hospeda artigos da área de Computação.

**Publicação em periódicos e revistas de acesso aberto**

Esta estratégia recomenda o desenvolvimento de uma nova geração de periódicos que usam os direitos de autor e outros instrumentos para garantir o acesso aberto permanente a todos os artigos que publicam. Os periódicos de acesso aberto permitem o acesso gratuito aos leitores e autorizam a reutilização da maioria dos seus conteúdos quase sem restrições. Vários periódicos têm sido criados com acesso aberto, por exemplo, a série *PeerJ*. O periódico *PeerJ Computer Science*[10] é o veículo voltado para a área de Computação.

O *Journal of Open Source Software* (JOSS)[11] é um periódico de acesso aberto voltado para a publicação de *pacotes de software de pesquisa*, sem taxas de processamento de artigos ou taxas de assinatura [Smith et al. 2017]. O JOSS possui um processo formal de revisão por pares para melhorar a qualidade do software submetido. Após a aceitação no JOSS, o artigo recebe um DOI e é listado no site do JOSS.

**Publicação de Artefatos de Pesquisa**

Além dos artigos científicos, repositórios de acesso aberto também são úteis para preservar e compartilhar *outros produtos e artefatos* gerados durante a pesquisa científica, permitindo que outros pesquisadores possam estudá-los e trabalhar com eles. Há diversos repositórios abertos internacionais, dentre eles, Digital Commons, DSpace e Wikimedia Commons [Abdo 2015]. Exemplos de repositórios abertos conhecidos pela comunidade brasileira são Zenodo, Figshare e OSF.

*Zenodo*[12] é um repositório aberto de uso geral, criado para apoiar os movimentos de acesso aberto e dados abertos na Europa. Pesquisadores podem depositar artigos, conjuntos de dados, *software de pesquisa*, relatórios e quaisquer outros artefatos digitais relacionados à pesquisa. Para cada envio, um identificador de objeto digital persistente (DOI) é criado, o que torna os itens armazenados facilmente citáveis. *Figshare*[13] permite a hospedagem de grandes quantidades de dados que aparecem em artigos online. O *Open*

[6] https://repositorio.ufba.br
[7] https://github.com/DSpace
[8] https://arxiv.org
[9] https://arxiv.org/archive/cs
[10] https://peerj.com/computer-science/
[11] https://joss.theoj.org
[12] https://zenodo.org
[13] https://figshare.com

*Science Framework (OSF)*[14] é uma ferramenta de código aberto para o gerenciamento de projetos que oferece suporte aos pesquisadores durante todo o ciclo de vida do projeto de pesquisa. É também uma ferramenta de colaboração, que dá suporte ao trabalho em equipe em projetos de forma privada ou torna todo o projeto e seus artefatos publicamente acessíveis para ampla divulgação.

### 3.2.4. Pesquisa Reprodutível Aberta

A *Pesquisa Reprodutível Aberta* é definida como "o ato de praticar Ciência Aberta e a provisão de oferecer aos usuários acesso gratuito a elementos experimentais para reprodução de pesquisas" [Bezjak et al. 2018]. A Reprodutibilidade é definida como a capacidade de investigadores independentes de tirar as mesmas conclusões de um experimento seguindo a documentação compartilhada pelos pesquisadores que originalmente realizaram a pesquisa. A reprodutibilidade na pesquisa científica aumenta a credibilidade da pesquisa.

Na pesquisa reprodutível aberta, os dados abertos fornecem os insumos necessários para que os pesquisadores validem e reproduzam os resultados uns dos outros. Entretanto, o acesso aos dados de pesquisa e artigos científicos não é mais suficiente para assegurar a reprodutibilidade na pesquisa. É importante definir e documentar os fluxos de trabalho (*workflows*) da pesquisa.

O pesquisador deve pensar em reprodutibilidade desde o início de sua pesquisa, documentando e compartilhando cada etapa do fluxo de trabalho – desde a coleta ou acesso aos dados, filtragem e limpeza, até a análise de dados – de modo a criar um roteiro ou guia para que ele e outros pesquisadores possam seguir. A documentação deve incluir e justificar decisões importantes, por exemplo, excluir certos valores ou ajustar certos parâmetros do modelo. O uso de software de *workflow* permite que cientistas automatizem e repliquem facilmente os fluxos de trabalho em cada etapa de coleta e análise de dados.

A iniciativa *Workflows Community*[15] compartilha recursos fornecidos pela comunidade de plataformas de software de coleta e análise de dados em pesquisa científica. Ferramentas como o Nextflow[16] orquestram, de maneira simplificada e aberta, *pipelines* de dados de maneira portável e escalável, com suporte a instalação em uma variedade de plataformas, incluindo computadores locais, infrastruturas como HPC e Kubernetes, e de Computação em Nuvem como AWS, Azure e Google Cloud. Outras soluções, por exemplo, o *Guix Workflow Language (GWL)*[17] oferecem extensões de apoio a computação científica para a linguagem declarativa de pacotes do GNU Guix, permitindo combinar pacotes Guix com workflows científicos.

Na Ciência Aberta, além dos dados abertos e fluxos de trabalho documentados, o software também é reconhecido como elemento experimental e parte fundamental do método científico.

---

[14] `https://osf.io`
[15] `https://workflows.community/`
[16] `https://nextflow.io`
[17] `https://guixwl.org`

### 3.2.5. Código Aberto

Na Ciência Aberta, *Código Aberto* é o termo usado para caracterizar software amplamente acessível para uso, seja como software livre, gratuito ou comercial [Bezjak et al. 2018]. "Aberto" pode ter dois significados diferentes: apenas o código-fonte está disponível em código aberto ou o software é desenvolvido de forma aberta [Hermann and Fehr 2022].

Projetos de Software Livre disponibilizam o código-fonte e são desenvolvidos de forma aberta, com licenças de software atribuídas que permitem ao usuário leitura, execução, adaptação e distribuição. Alguns pesquisadores consideram que o software livre é um pré-requisito para a Ciência Aberta [Flach and Kon 2021, Anzt et al. 2021].

Uma outra forma de compartilhar código aberto são os *notebooks*, fundamentados na Programação Letrada (*Literate Programming*) [Knuth 1984]). Um programa letrado é uma combinação de documentação e programa-fonte organizada de modo que possa ser lida por pessoas. Ferramentas de programação letrada extraem do arquivo uma documentação legível e um programa-fonte compilável. *Notebooks* Jupyter[18] são usados para análise exploratória e preparação de dados, treinamento e teste dos modelos. O código escrito em um *notebook* pode ser extraído e empacotado como um módulo ou biblioteca Python para posterior reutilização.

## 3.3. Software na Ciência

Na Ciência moderna, os dados de pesquisa (*research data*) dependem, de modo crescente, do ambiente de software em que foram criados, gerenciados, analisados e apresentados. Resultados digitais da pesquisa, como publicações científicas e conjuntos de dados abertos, não podem ser visualizados ou usados sem o software apropriado.

Muitas instituições, grupos e pesquisadores reconhecem que todo o software desenvolvido no contexto de uma pesquisa científica deve ser considerado um resultado de pesquisa [Jay et al. 2021] e que a boa prática científica exige que o software mencionado em publicações científicas seja mantido para reprodutibilidade e verificação de resultados científicos. O Manifesto do Código da Ciência (*Science Code Manifesto*)[19] destaca a importância do software usado ou desenvolvido *durante* a pesquisa científica:

> "Software é um produto de pesquisa essencial e o esforço para produzir, manter, adaptar, e fazer a curadoria do código deve ser reconhecido. O Software é parte de outras contribuições científicas vitais além de artigos publicados."

### 3.3.1. Software como Instrumento

Software é usado como parte integral de muitos instrumentos científicos, por exemplo, telescópios e microscópios. Outras vezes, o próprio software é o instrumento, gerando dados de pesquisa, validando dados de pesquisa, ou testando hipóteses. Isto inclui métodos computacionais ou modelos e simulações, por exemplo, modelos climáticos, modelos baseados em agentes em ciências sociais, simulação de hardware, dentre outros [Nieuwpoort and Katz 2023].

[18] `https://jupyter.org`
[19] `http://sciencecodemanifesto.org/`

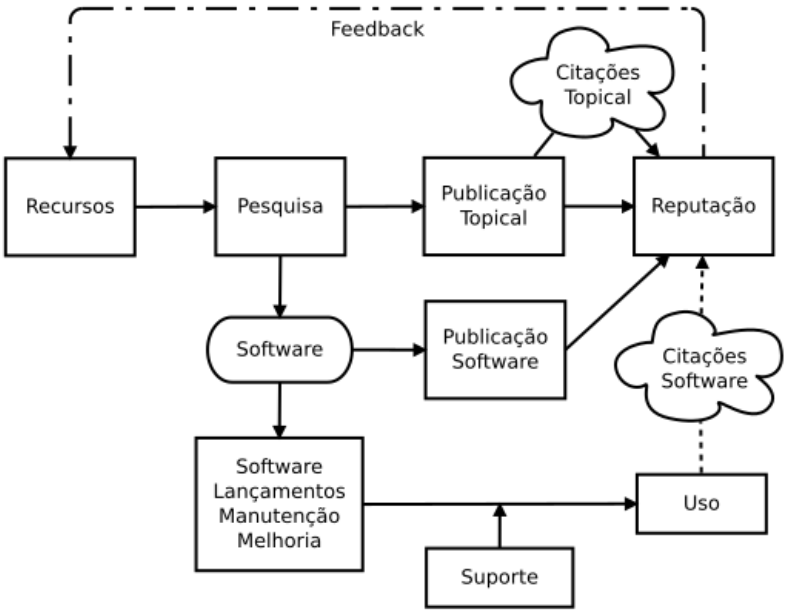



**Figura 3.2. Uma visão dos incentivos de reputação num contexto misto entre Ciência e práticas de software de pesquisa. [Howison and Herbsleb 2011]**

### 3.3.2. Software como Contribuição Científica

Software tornou-se parte fundamental de um ecossistema científico que engloba recursos, publicações [Howison and Herbsleb 2011] e possui a particularidade de se relacionar com o sistema econômico de reputação científica, especialmente com o seu modelo de publicações, influenciando e sendo influenciado diretamente pelo impacto de suas publicações [Howison et al. 2015].

A Figura 3.2 mostra que a *Pesquisa* conduzida por um ou mais pesquisadores pode gerar *Publicações científicas* com resultados para a área do conhecimento, e também pode produzir *Software* e *Publicação do software* [Howison et al. 2015]. As publicações científicas podem receber *Citações* dos pares, o que contribui para aumentar a *Reputação* acadêmica dos pesquisadores e, eventualmente, retroalimentar o sistema na forma de novos *Recursos* para a pesquisa.

A Figura 3.2 também relaciona *Software* a *Lançamentos, Manutenção e Melhoria* que exigem *Suporte* e levam à disponibilização do software para *Uso*. Entretanto, não há uma ligação direta e explícita entre a publicação do software e eventuais citações ao software. As setas tracejadas sugerem que o *Uso* do software pode gerar citações ao software e que estas podem influenciar de modo diferente (ou mesmo não influenciar) a reputação dos pesquisadores. Este problema pode ser explicado pela falta de padronização da citação de software e prática frequente de "citar" apenas uma URL que leva a um local onde o software pode ser encontrado.

A citação de software permite aos desenvolvedores receber reconhecimento e crédito pelo seu trabalho, além de facilitar a rastreabilidade e replicação dos resultados. Também contribui para a visibilidade e a colaboração na comunidade científica, fortalecendo a sustentabilidade técnica do software. No entanto, a importância do software na pesquisa ainda é subestimada, e o software é tratado como um recurso de apoio ou ferramenta, e não

como uma contribuição científica em si. No ecossistema da Ciência Aberta, o software de pesquisa deve ser citável e reconhecido como uma valiosa contribuição da pesquisa científica, tão importante quanto os artigos publicados, dados e metadados. No entanto, esforços e incentivos são necessários para que os créditos para o software de pesquisa se tornem mais específicos e rastreáveis na Ciência.

### 3.3.3. *Better Software, Better Research*

Pesquisadores sofrem uma intensa e constante pressão para publicar novos resultados rapidamente sem a devida cobrança sobre a necessidade de desenvolver código de qualidade. Como consequência, o software desenvolvido durante a pesquisa tende a não ser sustentável ou usável além da duração da pesquisa ou dos recursos que a financiam, e impede a comunidade científica de aproveitar plenamente o seu potencial impacto em pesquisas futuras.

Em 2016, inspirada pelo artigo *Better Software, Better Research* [Goble 2014], o workshop *Dagstuhl Perspectives* [Goble et al. 2016] reuniu ativistas, especialistas e interessados em discutir sobre o software produzido no contexto acadêmico e sua qualidade, na perspectiva de grupos de pesquisa que (i) consomem ou produzem software como saída do processo científico, ou (ii) consomem ou produzem software como um dos componentes dos métodos de pesquisa.

Dentre os resultados principais e com base em dados sobre o crescimento no uso de software em pesquisas científicas, o workshop identificou seis áreas no ecossistema de software de pesquisa que merecem atenção: *crédito ao software*, tornando-o rastreável nas pesquisas científicas; *herança cultural*, ou preocupação com as implicações das escolhas tecnológicas; *princípios FAIR para software* (discutidos na Seção 3.4.3); *financiamento* de atividades de desenvolvimento e manutenção de software; *políticas e e práticas* para incentivar a colaboração, a transparência e o compartilhamento de conhecimento entre os pesquisadores e instituições; e necessidade de *treinamentos* para pesquisadores e outros atores envolvidos com o desenvolvimento de software para uso na pesquisa.

Sobre treinamentos, o workshop recomendou que os mesmos devem cobrir diversos tópicos relacionados ao desenvolvimento de software e níveis de prática, e que cursos oferecidos por alguns institutos, por exemplo, *Carpentries*, devem ser estimulados e apoiados. Práticas centrais para manutenibilidade do software devem ser apresentadas, incluindo controle de versão, padrões de codificação, revisão de código, programação em par, testes, cobertura de código, integração contínua, refatoração e documentação. Outros tópicos também devem ser considerados, por exemplo, quão aberto é um software, se é desenvolvido por uma comunidade ou por um indivíduo, quão ativa é a manutenção do software (correção de *bugs* e criação de novas funcionalidades) ou disponibilidade de recursos diversos [Sufi et al. 2020].

## 3.4. Software de Pesquisa

### 3.4.1. Definição

O termo *software de pesquisa* foi criado para designar, de modo abrangente, o software *utilizado* durante a pesquisa científica, incluindo o software de terceiros usado para co-

leta, processamento e análise de dados [Allen et al. 2017]. No contexto de avaliação de qualidade do software de pesquisa, [Gruenpeter et al. 2021] adotam uma visão mais restrita, e consideram que sistemas operacionais, bibliotecas, dependências, pacotes e *scripts* utilizados na pesquisa científica, mas que não foram criados durante ou com uma intenção de pesquisa clara, são *software usado na pesquisa* mas *não são software de pesquisa*. No restante deste capítulo, adotaremos a visão de software de pesquisa de [Gruenpeter et al. 2021]:

*Software de Pesquisa é o software desenvolvido durante o processo de pesquisa e inclui (mas não está limitado a) código-fonte, algoritmos, scripts, fluxos de trabalho computacionais e executáveis.*

O *software de pesquisa* é parte integrante do ecossistema de pesquisa moderno, sendo amplamente utilizado nas Ciências e Engenharias para gerar resultados que servem como evidência em publicações científicas. Em geral, devido à complexidade em desenvolver software e ao conhecimento de domínio especializado necessário, os próprios pesquisadores desenvolvem o *software de pesquisa* ou estão intimamente envolvidos com o seu desenvolvimento [Carver et al. 2007].

Há várias preocupações relacionadas à *natureza* e à *qualidade* do *software de pesquisa*. Como artefato digital, ele pode assumir muitas formas, por exemplo, um *script* shell bash, com apenas 50 linhas para manipular e filtrar arquivos, um conjunto de *scripts* R de 100 linhas para análise de dados, 10.000 linhas de código Java para software de análise de dívida técnica ou 50.000 linhas de C++ para análise de imagens médicas.

Como produto de software, o *software de pesquisa* deve considerar alguns atributos de qualidade desejáveis. Por exemplo, é desejável que o *software de pesquisa* seja confiável, eficiente, fácil de usar, fácil compreender, testar, reparar, estender ou adaptar. Em especial, há uma preocupação com a sustentabilidade do *software de pesquisa* e sua influência na reprodutibilidade da pesquisa científica.

Neste capítulo, apresentamos aspectos relacionados à sustentabilidade do *software de pesquisa* e boas práticas para o desenvolvimento de software sustentável, a saber: registro e identificação do software, licenças de software e grau de abertura (*openness*), reconhecimento e citação do software, comunidade e usuários, longevidade e manutenibilidade, aderência aos princípios FAIR para software, avaliação do *software de pesquisa* e sua influência na reprodutibilidade científica.

### 3.4.2. Sustentabilidade

*Sustentabilidade* é um dos princípios norteadores da Ciência Aberta [UNESCO 2021] e define que a mesma deve se basear em práticas, serviços, infraestruturas e modelos de financiamento de longo prazo que garantam a participação igualitária dos indivíduos que produzem ciência originários de instituições e países menos privilegiados. As infraestruturas científicas abertas devem ser organizadas e financiadas com base em uma visão essencialmente sem fins lucrativos e de longo prazo, que aprimorem as práticas de Ciência Aberta e garantam o acesso permanente e irrestrito a todos, na medida do possível. No contexto da Ciência Aberta e sua dependência no código aberto (Seção 3.2.5), observa-se uma preocupação crescente com a sustentabilidade do software usado ou desenvolvido

durante a pesquisa, em especial, com a longevidade e disponibilidade do software, e seu impacto na reprodutibilidade científica.

O tema Sustentabilidade, inicialmente associado ao campo da Ecologia, agora faz parte dos interesses de outras áreas do conhecimento, ainda que adaptado às especificidades de cada uma. Na Ciência da Computação, a preocupação com a sustentabilidade emerge como um tópico importante em diversas subáreas, incluindo Inteligência Artificial, Computação de Alto Desempenho, Interação Humano-Computador, Computação Científica e Engenharia de Software [Venters et al. 2021]. Sustentabilidade é um desafio a ser enfrentado, não um problema a ser resolvido [Becker et al. 2015].

O *Manifesto de Karlskrona* para o Design Sustentável [Becker et al. 2015] define que sustentabilidade é, em sua essência, um conceito sistêmico, multi-facetado e deve ser entendida em um conjunto de cinco dimensões: recursos ambientais, social, bem-estar individual, prosperidade econômica e viabilidade técnica de longo prazo.

Neste capítulo, abordamos aspectos da dimensão técnica da sustentabilidade do software [Kehrer and Penzenstadler 2018] e, de forma complementar, elementos de sua dimensão social [de Souza 2023]. Não trataremos de aspectos da Engenharia de Software Verde [Mourão et al. 2018][20].

**Sustentabilidade de Software**

Na Engenharia de Software, há duas linhas de pesquisa voltadas para Sustentabilidade que se destacam: (i) Sustentabilidade de Software, e (ii) Engenharia de Software para Sustentabilidade (SE4S). A pesquisa sobre *Sustentabilidade de Software* tem como preocupação central a capacidade do software de perdurar ao longo do tempo, enquanto que a pesquisa relacionada a SE4S preocupa-se com sistemas intensivos em software, como integrar a sustentabilidade em seus processos de desenvolvimento de software e apoiar a sustentabilidade ambiental na ampla variedade de domínios em que o software é implantado [Venters et al. 2021].

No que se refere à sustentabilidade de software, a longevidade como expressão de tempo ("longo prazo") e a capacidade de manutenção são fatores-chave para sua compreensão [Venters et al. 2017]. A preocupação com a longevidade do software estende-se ao atributo de manutenibilidade, e ao modelo e processo de desenvolvimento de software adotados, que podem influenciar atributos relacionados à sustentabilidade. A manutenibilidade é reconhecidamente uma qualidade interna fundamental de sistemas de software [IEC 2014], enquanto que a sua relação com a sustentabilidade de software ainda é objeto de pesquisa na área. O termo sustentabilidade propriamente dito ainda não está bem definido neste contexto, deixando espaço para diferentes interpretações, e ainda há pouca evidência ou orientação sobre processos ou modelos de desenvolvimento voltados para a sustentabilidade de software [Venters et al. 2021].

De um ponto de vista puramente técnico, [Venters et al. 2021] define sustentabilidade de software como um requisito composto, não-funcional, de primeira classe, abran-

[20]A Engenharia de Software Verde leva em consideração práticas e arquitetura de software, design de hardware e *data center*, o mercado de eletricidade e mudanças climáticas, visando gerar menos emissões de gases de efeito estufa e reduzir produção de carbono de uma empresa [Mourão et al. 2018].

gendo as medidas de alguns conceitos centrais de atributos de qualidade de software, incluindo, no mínimo, manutenibilidade, extensibilidade e usabilidade. Na dimensão técnica, a sustentabilidade de software está relacionada às consequências de longo prazo de projetar, construir e entregar um projeto de software e se espalha por diversas áreas, dentre elas, qualidade de software e métricas, requisitos de software e arquitetura de software [Kehrer and Penzenstadler 2018, Venters et al. 2021]. Por fim, sustentabilidade de software diz respeito a assegurar que o software continue funcional para seus usuários ao longo do tempo, considerando também sua manutenção, inclusão de novos recursos, reparo de *bugs*, e adaptações a novos ambientes de software e hardware.

**Software de Pesquisa Sustentável**

O *Software de Pesquisa Sustentável* deve permanecer *disponível e funcional* para a comunidade científica durante períodos de tempo significativos. Não há uma resposta geral para a questão de quanto tempo o software precisa ser sustentado ou mantido. Este período pode depender da área de pesquisa, finalidade, função, frequência de uso, e da comunidade que o desenvolveu.

Sustentabilidade de *software de pesquisa* inclui o processo de desenvolvimento e manutenção de software para que o mesmo continue a cumprir seu propósito ao longo do tempo. Certamente, o *software de pesquisa* sustentável precisará ser atualizado com novas funcionalidades e correções de *bugs*, adaptado a novos ambientes computacionais, manter-se amigável aos seus usuários, tornar-se multiplataforma, ser testado e certificado. Garantir que o *software de pesquisa* continue executável é desafiador, sendo mais trabalhoso e caro do que um arquivamento simples de uma versão do software. As alterações necessárias para que o software permaneça executável devem garantir a confiabilidade nos resultados gerados pelas versões mais antigas do software. Se os resultados forem diferentes, justificativas objetivas devem ser fornecidas.

Há diversas maneiras de promover e investir na sustentabilidade do *software de pesquisa*, incluindo atração de desenvolvedores, suporte à comunidade de usuários, busca por financiamento ou até comercialização – todas válidas, desde que resultem na disponibilidade de longo prazo do software para a comunidade científica. Espera-se que, por meio da publicação de instruções, diretrizes e outras formas de ajuda e suporte, pesquisadores sejam capazes de decidir a forma de manter o seu software sustentável.

### 3.4.3. *FAIRness*

Os princípios FAIR [Wilkinson et al. 2016] foram especificados para melhorar o reuso de dados de pesquisa digitais, tornando-os mais fáceis de encontrar, acessíveis, interoperáveis e reutilizáveis (**F**indable, **A**ccessible, **I**nteroperable, **R**eusable). Tais princípios abordam a *forma* de fornecer artefatos para a comunidade científica, mas não tratam do conteúdo funcional ou da qualidade dos artefatos [Lamprecht et al. 2020]. Os *Princípios FAIR para Dados de Pesquisa* [Wilkinson et al. 2016], incluindo quatro princípios fundamentais e 15 princípios norteadores, estabelecem que os dados de pesquisa devem ser facilmente localizáveis, acessíveis, interoperáveis e reutilizáveis.

Segundo [Chue Hong et al. 2022], o *software de pesquisa* deve seguir os princípios FAIR usados para dados abertos, considerando-se que o software também é um

artefato de pesquisa digital e, como tal, deve ser facilmente localizável, acessível, interoperável e reutilizável. Os *Princípios FAIR para Software de Pesquisa* (FAIR4RS) foram definidos a partir de uma reformulação dos princípios FAIR originais para dados abertos [Lamprecht et al. 2020, Chue Hong et al. 2022, Barker et al. 2022]. É importante destacar que, diferentemente dos dados, o software não é um artefato estático e só pode ser (re)utilizado se for sustentável [Lamprecht et al. 2020]. A Tabela 3.1 apresenta a versão mais recente dos princípios FAIR para *software de pesquisa*.

*Projetos de Software Livre* seguem os princípios FAIR. Software livre pode ser localizado em repositórios com base em identificadores e descritores, utilizando diversos critérios como palavras-chave, linguagem de programação, versão do software, entre outros. A acessibilidade é encorajada em software disponível em repositórios abertos, com licenças de compartilhamento explícitas e bem definidas e documentação associada. A definição de interfaces de programação, formatos de entrada/saída e uso de padrões promovem a interoperabilidade e o reúso por vários grupos de pesquisa. Nessa perspectiva, as práticas usadas no modelo de desenvolvimento de software livre podem ser adotadas no desenvolvimento e evolução de *software de pesquisa* [Flach and Kon 2021].

**Tabela 3.1. Princípios FAIR para Software [Barker et al. 2022].**

| **Princ.** | **Descrição** |
|---|---|
| F: | Software, and its associated metadata, is easy for both humans and machines to find. |
| F1 | Software is assigned a globally unique and persistent identifier. |
| F1.1 | Software components representing granularity levels are assigned distinct identifiers. |
| F1.2 | Different versions of the software are assigned distinct identifiers. |
| F2 | Software is described with rich metadata. |
| F3 | Metadata clearly and explicitly include the identifier of the software they describe. |
| F4 | Metadata are FAIR, searchable and indexable. |
| A: | Software, and its metadata, is retrievable via standardised protocols. |
| A1 | Software is retrievable by its identifier using a standardised communications protocol. |
| A1.1 | The protocol is open, free, and universally implementable. |
| A1.2 | The protocol allows for authentication and authorization procedure, where necessary. |
| A2 | Metadata are accessible, even when the software is no longer available. |
| I: | Software interoperates with other software by exchanging data and/or metadata, and/or through interaction via application programming interfaces (APIs), described through standards. |
| I1 | Software reads, writes and exchanges data in a way that meets domain-relevant community standards. |
| I2 | Software includes qualified references to other objects. |
| R: | Software is both usable (can be executed) and reusable (can be understood, modified, built upon, or incorporated into other software). |
| R1 | Software is described with a plurality of accurate and relevant attributes. |
| R1.1 | Software is given a clear and accessible license. |
| R1.2 | Software is associated with detailed provenance. |
| R2 | Software includes qualified references to other software. |
| R3 | Software meets domain-relevant community standards. |

### 3.4.4. Reprodutibilidade

A *Reprodutibilidade* é um dos fundamentos do método científico e um princípio norteador da Ciência Aberta [UNESCO 2021]. A boa prática científica exige que os artefatos de pesquisa mencionados em publicações científicas sejam mantidos e fiquem disponíveis para escrutínio dos pares, reprodução independente e verificação de resultados.

Para o *software de pesquisa*, a submissão ou publicação de um artigo é um dos momentos em que a versão do software usada no estudo precisa ser identificada, documentada e lançada. Se houver modificações no *software de pesquisa*, elas precisam ser registradas, usando algum esquema de nomenclatura que identifique o <*software*>, a <*versão*> e o <*lançamento*>. Em geral, a versão refere-se a mudança estratégica durante a evolução do software e o lançamento refere-se a mudanças simples de serviço.

A recomendação para que pesquisadores compartilhem e permitam o acesso aos dados descritos em suas publicações científicas, não garante a reprodutibilidade da pesquisa. Na prática, pesquisadores devem compartilhar o código de todo o *software de pesquisa* desenvolvido e fluxos de trabalho de suas pesquisas para assegurar a reprodutibilidade. Na maioria dos casos, o arquivamento de baixo custo do software, acompanhado de documentação clara deve ser suficiente.

Diretrizes para o desenvolvimento sustentável de *software de pesquisa* também podem contribuir para a condução de uma pesquisa científica reprodutível, apresentando princípios e boas práticas da engenharia de software. As implementações das diretrizes podem variar entre os diferentes domínios científicos, mas as implementações devem ser públicas, abertas para discussão entre os pesquisadores e continuamente adaptadas em reposta às mudanças tecnológicas e nos domínios.

### 3.4.5. Avaliação de *software de pesquisa*

Consideramos a avaliação do *software de pesquisa* sob duas perspectivas: (1) sustentabilidade ou direcionada a sua longevidade, e (2) aderência aos princípios FAIR (*FAIRness*) ou direcionada ao grau de abertura (*openness*) do *software de pesquisa*.

#### Sustentabilidade

A avaliação da sustentabilidade do *software de pesquisa* busca determinar se o mesmo é sustentável com base em critérios relevantes para o ecossistema científico. Em geral, a avaliação não gera um resultado *é sustentável/não é sustentável*, mas tende a valorizar pontos fortes do software e identificar pontos para sua melhoria. A avaliação pode considerar atributos ou práticas usadas no desenvolvimento do software para prover uma visão geral ou detalhada da sustentabilidade do *software de pesquisa*.

O Instituto de Sustentabilidade de Software (SSI) fornece um serviço de avaliação quantitativa do software baseada em critérios relacionados a sustentabilidade, manutenibilidade e usabilidade[21]. A avaliação é feita com base em respostas dadas a um conjunto de perguntas simples, seguidas por explicações sobre a importância de algumas práticas e recomendações. O conjunto de perguntas é aplicável para *software de pesquisa*.

[21]O questionário do SSI para avaliação de sustentabilidade está disponível em `http://www.software.ac.uk/online-sustainability-evaluation`.

**Tabela 3.2. Avaliação de sustentabilidade do *software de pesquisa* baseada em práticas.**

| P | Descrição | Atende? | Comentário |
|---|---|---|---|
| P1 | O software está hospedado em um repositório público | | |
| P2 | O software utiliza controle de versão | | |
| P3 | O software adota explicitamente uma licença | | |
| P4 | O software está registrado e apresenta um DOI | | |
| P5 | A estrutura de arquivos do projeto de software comunica a finalidade de seus elementos | | |
| P6 | O software usa formato de dados e interfaces padronizadas | | |
| P7 | A documentação apresenta uma visão geral do software | | |
| P8 | O software possui testes | | |
| P9 | O código é revisado antes de ser publicado | | |
| P10 | O projeto de software utiliza rastreador de tarefas e *bugs* | | |
| P11 | Tarefas repetitivas são automatizadas | | |
| P12 | Há integração e implantação contínuas | | |
| P13 | Há lançamento de versões do software | | |
| P14 | Há evidência de uma comunidade (presente ou futuro) | | |
| P15 | O software é divulgado para a comunidade acadêmica | | |
| P16 | Há uma forma recomendada para citação do software | | |

Por exemplo, para uma resposta negativa para a pergunta "*Seu software de pesquisa está disponível como um pacote que pode ser instalado sem compilar?*", a avaliação oferece a recomendação geral "Construir software pode ser complicado e demorado. Fornecer seu *software de pesquisa* como um pacote que pode ser implantado sem compilar pode economizar tempo e esforço dos usuários, especialmente se não forem desenvolvedores de software. Idealmente, deve-se observar e testar se o *software de pesquisa* é compilado e executado em diversas plataformas para as quais ele oferece suporte, para então prover pacotes para sua distribuição."

Em nossa pesquisa, a avaliação de sustentabilidade é simples, baseada em práticas, e busca determinar se um *software de pesquisa* foi desenvolvido seguindo boas práticas que potencialmente o qualificariam como software sustentável. As práticas usadas em nossas avaliações estão relacionadas a identidade e disponibilidade do software, adoção de licenças, controle de versão, documentação, estrutura do código-fonte, testes, política de contribuidores e suporte, entre outras. O resultado de cada avaliação é sintetizado em uma tabela e apresentado em um relatório simples. A Tabela 3.2 mostra práticas e aspectos que consideramos na avaliação da sustentabilidade de *software de pesquisa*.

### *FAIRness*

A avaliação de *FAIRness* do *software de pesquisa* pode ser definida como um processo de avaliação do seu grau de aderência aos princípios FAIR adaptados para software. É importante lembrar que os princípios FAIR não são prescritivos, mas buscam oferecer uma visão para melhorar o compartilhamento e a reutilização de dados ou software por pessoas e máquinas [Wilkinson et al. 2016, Chue Hong et al. 2022]. Ainda assim, a avaliação de *FAIRness* para software pode servir para apoiar a decisão de usuários e desenvolvedores sobre uma eventual adoção do *software de pesquisa* em seu projeto de pesquisa.

No estudo apresentado neste capítulo, consideramos a versão mais recente dos princípios FAIR para software [Barker et al. 2022].

## 3.5. Desenvolvimento de Software de Pesquisa

Em geral, o desenvolvimento de software de pesquisa exige conhecimento específico sobre o domínio do estudo sendo realizado, por exemplo, entender como o DNA genômico se transforma em cristais de proteína, ou estar familiarizado com os meandros da dinâmica dos fluidos, ou saber como resolver 20 equações diferenciais parciais simultâneas [Segal and Morris 2008]. Isto explica a grande participação dos cientistas no desenvolvimento de software de pesquisa.

No Reino Unido, um estudo pioneiro de [Hettrick et al. 2014] envolvendo todas as áreas da Ciência, mostrou que 56% dos cientistas estavam envolvidos no desenvolvimento de software acadêmico [Hettrick et al. 2014]. Outros estudos em grupos específicos mostraram números ainda mais expressivos. Na área de Astronomia, por exemplo, 90% dos cientistas desenvolvem software de pesquisa [Momcheva and Tollerud 2015]. No entanto, a maior parte dos cientistas não havia recebido treinamento sobre conceitos e boas práticas de desenvolvimento de software.

Em geral, os pesquisadores que desenvolvem o seu *software de pesquisa* não testam ou documentam os seus projetos de software, e não seguem práticas básicas de desenvolvimento, como escrever código legível, revisar código, usar controle de versão ou testes unitários [Wilson et al. 2017]. A falta de conhecimento sobre a natureza do software, conceitos e práticas de desenvolvimento de software pode ocasionar sérios erros computacionais em conclusões centrais da literatura acadêmica, gerando retrabalho para retratar tais erros nas mais diversas áreas da Ciência [Merali 2010]. Além disso, dados podem ser perdidos, análises podem consumir mais tempo que o necessário e a pesquisa pode ter sua eficiência comprometida se pesquisadores não desenvolverem e trabalharem com *software de pesquisa* de qualidade [Wilson et al. 2017]. Por fim, o desconhecimento sobre práticas de desenvolvimento colaborativo e aberto pode causar um impacto negativo na visibilidade do *software de pesquisa*, na capacidade de ser encontrado e compartilhado [Howison and Herbsleb 2013, Katz 2014] e em sua sustentabilidade.

### 3.5.1. Ciclo de Vida do Software

O modelo de ciclo de vida do software proposto por Rajlich [Rajlich and Bennett 2000] descreve o software como um produto em constante evolução e serve para contextualizar os estágios de um *software de pesquisa* sustentável. A Figura 3.3 apresenta as etapas ou estágios em que o software pode se encontrar ao longo de sua existência.

Na etapa de Desenvolvimento inicial (*Initial development*), desenvolvedores implementam a primeira versão funcional do software. Se o desenvolvimento inicial for bem-sucedido, o software entra no estágio de Evolução (*Evolution*), quando ocorrem mudanças iterativas, modificações e exclusões de funcionalidade. O modelo de Rajlich destaca que, em determinados intervalos, *uma versão do software é liberada para a comunidade*. Na etapa de Serviço (*Servicing*), desenvolvedores fazem pequenos reparos de defeitos e mudanças funcionais simples. Na etapa de *Phaseout*, o software não recebe manutenção, mas continua disponível. No estágio de Fechamento (*Closedown*), o software

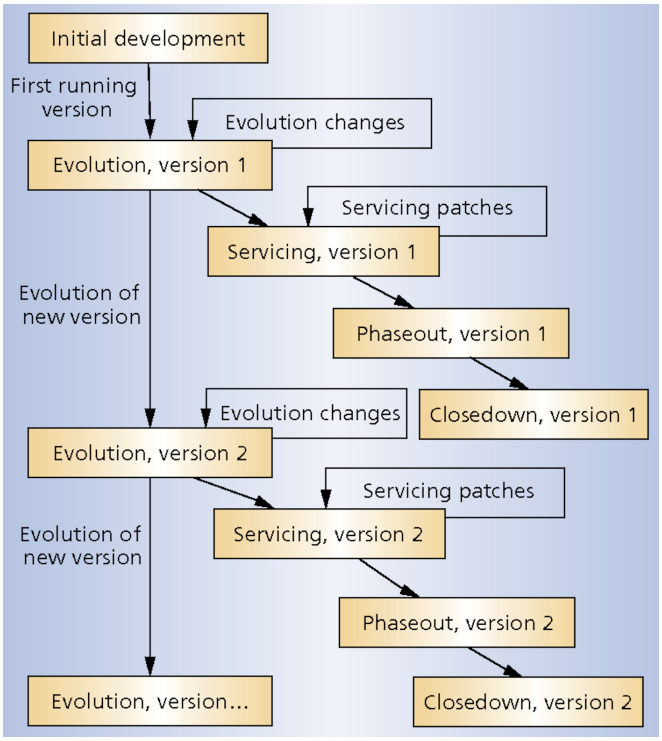



**Figura 3.3.** ***O modelo de Rajlich com versões enfatiza a natureza evolutiva do desenvolvimento de software. Fonte: [Rajlich and Bennett 2000].***

deixa de existir e os usuários podem ser redirecionados para outro software.

No contexto de desenvolvimento de *software de pesquisa*, deve-se considerar que ele se insere em um projeto de pesquisa que inclui *dados e software*. No início do desenvolvimento do *software de pesquisa* deve-se decidir sobre a licença do software, onde o software será armazenado (por exemplo, GitHub, GitLab, etc.), tipo de sistema de versionamento e padrões de entrada e saída. O lançamento de uma versão do *software de pesquisa* e sua liberação para a sua comunidade podem ser motivados pela publicação de um artigo científico. O texto do artigo deve mencionar claramente a versão do *software de pesquisa* utilizada.

Também é essencial considerar a continuidade do *software de pesquisa* ao longo do tempo e o suporte à reprodutibilidade, especialmente quando o software for usado na pesquisa publicada em artigos científicos. Ao contrário de outros tipos de software, a etapa de Fechamento (*Closedown*) não deveria ser considerada em *software de pesquisa* para permitir a reprodutibilidade dos resultados divulgados nos artigos publicados. Manter o *software de pesquisa* vivo, disponível e acessível é fundamental para que outros cientistas possam reproduzir e verificar os resultados da pesquisa.

### 3.5.2. Pessoas

Considerando que muitos problemas de manutenção de software não são problemas técnicos, mas problemas relacionados às pessoas [Versen 2020], percebemos que a dimensão social desempenha um papel importante na evolução do software, influenciando tanto o processo quanto o resultado final. As pessoas interessadas no desenvolvimento, utilização

e manutenção de software têm um impacto significativo em sua evolução. A interação e colaboração entre os membros da equipe de desenvolvimento e os usuários contribuem com suas perspectivas e experiências, influenciando as decisões sobre novas funcionalidades, correções de *bugs* e melhorias em geral. Além disso, as demandas e expectativas dos usuários, assim como mudanças em suas necessidades e preferências podem exigir adaptações e atualizações do software para atender a essas demandas.

O caso específico da evolução de um *software de pesquisa* apresenta características distintas. O *software de pesquisa* está inserido em um ecossistema científico, onde os pesquisadores buscam avançar o conhecimento em suas áreas. Nesse contexto, sua evolução está intimamente ligada à evolução da própria pesquisa e ao surgimento de novas abordagens e técnicas, exigindo atualizações e modificações no software. Além disso, a estrutura do grupo de pesquisa também influencia a evolução do software. À medida em que a pesquisa avança, novos membros ingressam no grupo, pesquisadores saem e a equipe se reconfigura. Tais mudanças na composição do grupo demandam mecanismos para transmitir o conhecimento legado e práticas estabelecidas.

Tornar um *software de pesquisa* disponível publicamente e aceitar contribuição de pessoas externas ao grupo de pesquisa pode resultar em uma comunidade maior de desenvolvedores envolvidos, facilitando o desenvolvimento mais rápido e eficiente, incluindo correções de *bugs* e implementação de novos recursos de forma colaborativa. No entanto, a coordenação de um *software de pesquisa* aberto pode exigir esforço adicional para estabelecer processos de governança e revisão de contribuições, para garantir que estas sejam de alta qualidade e estejam alinhadas com os objetivos do projeto. Além disso, a abertura precoce do *software de pesquisa* pode expor pesquisas em andamento antes que estejam prontas para a publicação, o que pode comprometer a confidencialidade e ineditismo dos resultados.

### 3.5.3. Práticas

Há alguns anos a comunidade científica já reconhece ser essencial que cientistas, pesquisadores e estudantes sejam capazes de aprender e adotar um novo conjunto de habilidades e metodologias relacionadas a software [Katz et al. 2016]. Alguns pesquisadores já abraçaram boas práticas de desenvolvimento de software e, em particular, uma classe especializada de desenvolvedores de software, chamados de *Engenheiros de Software de Pesquisa* (*Research Software Engineers*), está surgindo em ambientes acadêmicos como parte integrante de grupos de pesquisa bem-sucedidos [Katz et al. 2016]. O uso consistente de boas práticas é apoiado por tecnologias e ferramentas conhecidas e tradicionalmente usadas por engenheiros de software.

## 3.6. Práticas para o Desenvolvimento de Software de Pesquisa Sustentável

Projetos de software bem organizados adotam um modelo de desenvolvimento que resulta em código de alta qualidade que se adapta rapidamente a diferentes situações. Várias *práticas* estabelecidas e usadas pelas comunidades de software livre são reconhecidas como contribuições importantes para a manutenção de *software de pesquisa* de alta qualidade. Nesta seção, apresentamos dezesseis boas práticas para uso em projetos de *software de pesquisa*, descritas e ilustradas no contexto de projetos hospedados no GitHub. Parte do

jargão técnico usado pelo GitHub não foi traduzido e alguns termos não foram definidos. Termos e suas definições podem ser encontrados no *Glossário do GitHub*[22].

### 3.6.1. Práticas Básicas

#### P1. Hospedagem do projeto

É recomendável hospedar e compartilhar o *software de pesquisa* em um repositório público, exceto se houver questões de confidencialidade ou estratégicas relacionadas ao projeto de pesquisa (por exemplo, software implementa um algoritmo original, ainda não compartilhado amplamente com a comunidade científica). A hospedagem em repositório público pode ser estendida para todos os ativos da pesquisa, por exemplo, dados, fluxos de trabalho e relatórios.

A *hospedagem pública do software* é fundamental para promover a reprodutibilidade, evitar a redundância (por exemplo, cópias do software em diversos repositórios locais), e estimular o compartilhamento entre pares da comunidade acadêmica. A hospedagem pública do software e o acesso aos dados e fluxos de trabalho, permitem que os resultados da pesquisa possam ser verificados, replicados, e reproduzidos, fortalecendo a confiabilidade e a transparência dos estudos. Além disso, o *software de pesquisa* hospedado torna-se localizável e acessível, facilitando seu uso ou reuso. Se o código for aberto, outros pesquisadores podem se beneficiar dele, evitando o retrabalho e construindo sobre uma base já estabelecida. Isso promove a colaboração, a troca de conhecimentos e a aceleração do progresso científico.

**Exemplo.** O *software de pesquisa* `flosssearch` estava hospedado em uma pasta local no computador de um pesquisador. Por segurança, o pesquisador realizava backup manual periodicamente. Ao finalizar a implementação da primeira funcionalidade do software, o pesquisador decidiu que o projeto precisava ser hospedado adequadamente e compartilhado com outros membros do grupo de pesquisa. O software `flosssearch` foi colocado em um repositório público no GitHub, uma plataforma baseada na web onde os usuários podem hospedar repositórios, compartilhar e promover a colaboração em projetos de software. O repositório foi compartilhado com dois pesquisadores do grupo.

Ao ser hospedado no GitHub, o software recebeu uma URL, um nome oficial e a sua identidade foi definida, de forma clara e única. O nome oficial público atribuído ao *software de pesquisa* é `flosssearch`. Além disso, o software `flosssearch` passou a contar com um serviço de backup, considerando que os arquivos do projeto e todo o seu histórico são armazenados no servidor remoto e nos repositórios locais dos pesquisadores.

#### P2. Controle de versão

Controle de versão é a prática de rastrear e gerenciar alterações no código de um software. Os sistemas de controle de versão são ferramentas que permitem que várias versões do mesmo *software de pesquisa* sejam mantidas e, possivelmente, referenciadas por experimentos de pesquisa e artigos científicos. Um sistema de controle de versão também permite o acompanhamento das atividades realizadas, fornece um registro das alterações e permite que desenvolvedores e usuários do *software de pesquisa* acompanhem o seu

[22] `https://docs.github.com/en/get-started/quickstart/github-glossary`

desenvolvimento e evolução, tornando a pesquisa mais aberta e reprodutível. Ao usar o controle de versão, o pesquisador nunca perde as versões anteriores do *software de pesquisa*, e erros podem ser revertidos. Adicionalmente, o versionamento do software também promove a colaboração entre pesquisadores.

**Exemplo.** A plataforma GitHub utiliza o Git[23], um sistema de controle de versão distribuído gratuito e de código aberto. Ao ser colocado no GitHub, com um nome e repositório associado, o *software de pesquisa* `flosssearch` recebeu suporte para controle de versão distribuído.

### P3. Licenças de Software

A licença de software escolhida é essencial para o *software de pesquisa* disponível publicamente. O financiamento do projeto pode terminar após um certo período, e os mantenedores podem terminar suas pesquisas ou mudar de área de interesse. Assim, para garantir a disponibilidade contínua do projeto, os desenvolvedores precisam chegar a um acordo formal, ou seja, uma licença de software em que os termos de uso sejam explícitos.

Recomenda-se que projetos de *software de pesquisa* de código aberto sejam lançados sob uma licença compatível com a GNU General Public License - GPL[24], a licença OSS mais popular e amplamente utilizada. Outras licenças de software de código aberto e suas descrições podem ser encontradas no site da Open Source Initiative (OSI)[25].

O GitHub permite que o usuário escolha uma licença de software para o seu repositório e cria automaticamente o arquivo *LICENSE* com uma cópia da licença escolhida. Entretanto, apenas colocar uma cópia da licença em um arquivo em seu repositório não declara explicitamente que o código no repositório pode ser usado sob tal licença. Sem uma declaração explícita, não fica claro se as permissões da licença se aplicam a qualquer arquivo fonte no repositório.

A Figura 3.4 mostra um aviso que o desenvolvedor deve incluir no início de cada arquivo fonte para declarar a licença e a exclusão da garantia. Cada arquivo deve ter pelo menos a linha "copyright" e indicação sobre local onde o aviso completo pode ser encontrado. A falta de uma licença claramente definida pode gerar incerteza legal e dificultar a contribuição e o compartilhamento do software, já que os colaboradores podem ficar inseguros sobre seus direitos e obrigações, o que pode limitar a adoção e colaboração.

**Exemplo.** Ao ser colocado em um repositório público no GitHub, o *software de pesquisa* `flosssearch` tornou-se visível, acessível e utilizável por terceiros, sendo necessária a escolha de uma licença de código aberto. Os pesquisadores escolheram a licença GNU GPL 3.0. Além do arquivo LICENSE, os arquivos com código-fonte do projeto possuem uma breve descrição sobre o seu propósito, a linha "copyright" e indicação sobre a licença de software escolhida e onde o texto completo pode ser encontrado (Figura 3.5).

[23] `https://git-scm.com`
[24] `https://www.gnu.org/licenses/gpl-3.0.html`
[25] `https://opensource.org/licenses/`

<*one line to give the program's name and a brief idea of what it does.*>

Copyright (C) <*year*> <*name of author*>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

**Figura 3.4. Declaração explícita sobre uso de licença GPL e exclusão da garantia.**

`flossearch` is a web application designed with the goal of supporting instructors and students in the selection of OSS projects for software engineering education.

Copyright (C) *2021 Moara Brito*

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License 3.0 <https://www.gnu.org/licenses/>.

**Figura 3.5. Declaração sobre uso de licença GPL para inclusão nos arquivos do `flosssearch`.**

### P4. Registro de Software

O *software de pesquisa* deve ser publicado formalmente para satisfazer os princípios FAIR, promover a sustentabilidade de software e permitir a citação de software. Os repositórios de publicação dão suporte ao registro do software e fornecem versões de software publicadas com identificadores únicos e persistentes, como um DOI.

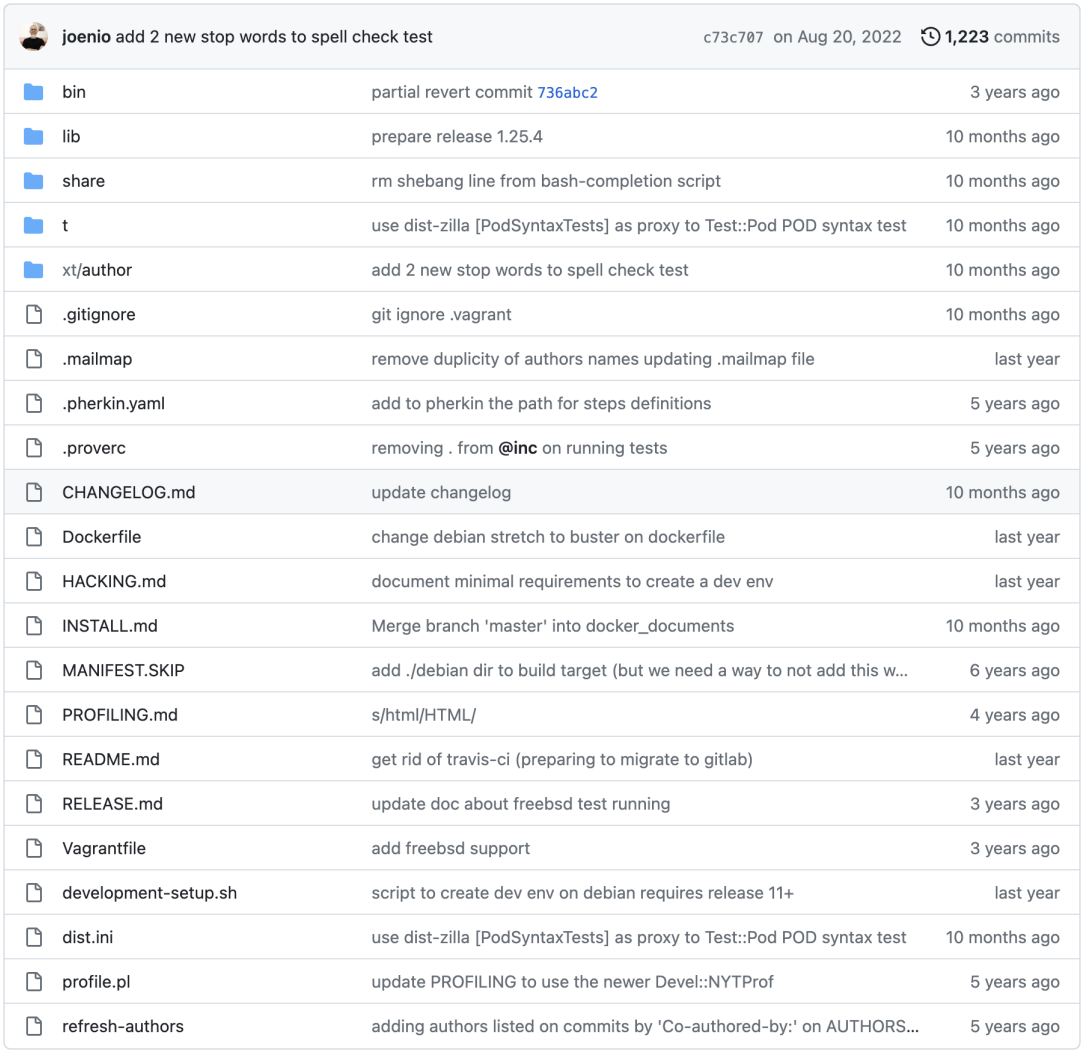
Há várias formas de associar um DOI a um repositório GitHub. Zenodo[26] e Figshare[27] podem ser utilizados para fazer o registro do DOI e arquivar versões especificas do software. Há também o projeto *The Research Software Directory*[28] desenhado para centralizar metadados de software de pesquisa e mostrar os seus impactos na pesquisa e na sociedade, estimulando o reuso de software e encorajando a citação de maneira apropriada para garantir que pesquisadores e engenheiros de *software de pesquisa* recebam créditos por seus trabalhos.

**Exemplo.** O *software de pesquisa* `Parcels`[29] (**P**robably **A** **R**eally **C**omputationally **E**fficient **L**agrangian **S**imulator) é um conjunto de classes e métodos em Python para criar simulações de rastreamento de partículas, como plâncton, plástico e peixes. O `Parcels` está registrado no Research Software Directory[30].

[26] `https://zenodo.org`
[27] `https://figshare.com`
[28] `https://research-software-directory.org`
[29] `https://github.com/OceanParcels/parcels`
[30] `https://research-software-directory.org/software/parcels`

### 3.6.2. Organização do Projeto

O suporte à reprodutibilidade na pesquisa que utiliza um *software de pesquisa* requer que seus artefatos estejam bem organizados em um repositório aberto.



| | | |
|---|---|---|
| **joenio** add 2 new stop words to spell check test | c73c707 on Aug 20, 2022 | **1,223** commits |
| bin | partial revert commit 736abc2 | 3 years ago |
| lib | prepare release 1.25.4 | 10 months ago |
| share | rm shebang line from bash-completion script | 10 months ago |
| t | use dist-zilla [PodSyntaxTests] as proxy to Test::Pod POD syntax test | 10 months ago |
| xt/author | add 2 new stop words to spell check test | 10 months ago |
| .gitignore | git ignore .vagrant | 10 months ago |
| .mailmap | remove duplicity of authors names updating .mailmap file | last year |
| .pherkin.yaml | add to pherkin the path for steps definitions | 5 years ago |
| .proverc | removing . from **@inc** on running tests | 5 years ago |
| CHANGELOG.md | update changelog | 10 months ago |
| Dockerfile | change debian stretch to buster on dockerfile | last year |
| HACKING.md | document minimal requirements to create a dev env | last year |
| INSTALL.md | Merge branch 'master' into docker_documents | 10 months ago |
| MANIFEST.SKIP | add ./debian dir to build target (but we need a way to not add this w... | 6 years ago |
| PROFILING.md | s/html/HTML/ | 4 years ago |
| README.md | get rid of travis-ci (preparing to migrate to gitlab) | last year |
| RELEASE.md | update doc about freebsd test running | 3 years ago |
| Vagrantfile | add freebsd support | 3 years ago |
| development-setup.sh | script to create dev env on debian requires release 11+ | last year |
| dist.ini | use dist-zilla [PodSyntaxTests] as proxy to Test::Pod POD syntax test | 10 months ago |
| profile.pl | update PROFILING to use the newer Devel::NYTProf | 5 years ago |
| refresh-authors | adding authors listed on commits by 'Co-authored-by:' on AUTHORS... | 5 years ago |

**Figura 3.6. Estrutura de arquivos da ferramenta `Analizo`.**

### P5. Estrutura

É importante ter uma estrutura de arquivos que comunica a finalidade dos elementos dentro de um *software de pesquisa*, separando os interesses em uma hierarquia de pastas e usando nomes auto-explicativos, facilitando a compreensão de todos que tenham interesse revisitar, revisar e desenvolver o software.

**Exemplo.** A Figura 3.6 apresenta a estrutura de arquivos da ferramenta de análise estática `Analizo`[31][Terceiro et al. 2010], desenvolvida como *software de pesquisa* no contexto de uma pesquisa de doutorado. Pode-se observar que há pastas e arquivos com nomes auto-explicativos, por exemplo, *bin*, *lib* e *share*, INSTALL.md, README.md.

[31] `https://github.com/analizo/analizo`

### P6. Padronização

A padronização e o uso de protocolos de comunicação entre sistemas tem uma grande importância no software de pesquisa. A partir da adoção de padrões e protocolos bem definidos, os diferentes sistemas podem interagir de forma consistente, facilitando a integração e desencoraja a dependência de fornecedores específicos. Além disso, a padronização simplifica a interoperabilidade e a reutilização de componentes de software, permitindo que os pesquisadores desenvolvam seus trabalhos sobre trabalhos existentes e tornem mais rápido o desenvolvimento de novas soluções.

**Exemplo.** O software `MoSyn` é um *software de pesquisa* que se preocupa com padronização pois depende de um software proprietário, o MATLAB[32]. A Seção 3.7 apresenta uma avaliação do `MoSyn` com base nas práticas para o desenvolvimento adotadas.

### P7. Documentação

A documentação frequente e contínua é uma prática recomendada para manter guias do usuário, manuais e outros documentos relevantes atualizados em relação à versão mais recente do software, facilitando o entendimento e colaboração. Para o *software de pesquisa*, a documentação é um pré-requisito fundamental para reuso e reprodução de estudos [Chue Hong et al. 2022]. [Hermann and Fehr 2022] analisaram o estado-da-prática sobre documentação de *software de pesquisa*. Eles reportaram que, ainda que alguns guias com boas práticas científicas valorizem e recomendem a prática de documentar explicitamente [Forschungsgemeinschaft 2022], a documentação do *software de pesquisa* ainda é considerada inadequada [Wilson et al. 2017, Chue Hong et al. 2022].

A documentação de um projeto só é útil se está atualizada. Por isso, uma questão importante em um projeto de software é a sincronização entre o desenvolvimento do software e a documentação dele, garantindo que a documentação representa a versão mais atualizada do software. Para conseguir essa sincronização é essencial integrar o processo de atualização da documentação no ciclo de desenvolvimento do software. Além disso, é possível automatizar a geração de documentação a partir de comentários no código-fonte, reduzindo o esforço necessário para manter a documentação atualizada.

Em geral, geradores de documentação são usado para interfaces de programação de aplicativos (APIs), destinadas a um público de desenvolvedores. *Doxygen*[33] é uma ferramenta que extrai documentação a partir do código-fonte e de outros arquivos e gera uma versão em formato estruturado, como HTML, PDF ou LaTeX. Doxygen é a ferramenta padrão de fato para gerar documentação para C++, mas também suporta outras linguagens de programação populares como C, Objective-C, C#, PHP, Java e Python.

Uma documentação precisa ser clara e considerar os diferentes interessados no *software de pesquisa*. A documentação precisa atender às necessidades específicas de cada grupo, usando linguagem e terminologia que ajudem cada público a compreender e utilizar as informações de maneira eficaz.

A *documentação para desenvolvedores* deve abordar aspectos técnicos, como arquitetura, APIs, dependências e configurações, incluindo descrições detalhadas de como

[32] `https://www.mathworks.com/products/matlab.html`
[33] `https://www.doxygen.nl`

utilizar as funcionalidades do software. É importante que sejam fornecidas orientações passo-a-passo sobre instalação, configuração, execução e dependências necessárias para o ambiente de desenvolvimento do software. Para facilitar a contribuição, é essencial descrever conceitos fundamentais, padrões de codificação e boas práticas de desenvolvimento. Testes de software também oferecem uma forma de documentar a funcionalidade do software por meio da qual desenvolvedores podem ter uma visão mais detalhada sobre o funcionamento interno do software e integração com outros sistemas.

Na *documentação para pesquisadores* que não são desenvolvedores, deve-se utilizar uma linguagem acessível, evitando jargões técnicos e explicando conceitos de forma simples e compreensível. A descrição da utilização deve destacar as funcionalidades e benefícios do software. Para facilitar o uso, podem ser apresentados exemplos práticos de como o *software de pesquisa* pode ser aplicado em diferentes cenários e incluir uma seção de perguntas frequentes para responder às dúvidas mais comuns dos usuários. Dessa forma, a documentação para os pesquisadores que não são desenvolvedores pode ajudá-los a entender como utilizar o software de forma prática para atingir seus objetivos de pesquisa, sem detalhes técnicos desnecessários.

Para todos os tipos de usuários, recomenda-se que, ao publicar o *software de pesquisa*, o desenvolvedor disponibilize uma página de boas-vindas e descrição do software. Isso pode ser feito facilmente com um arquivo *README* colocado no diretório raiz do projeto. Uma boa descrição sintetiza as funcionalidades do *software de pesquisa* e permite que outros pesquisadores possam decidir se querem usá-lo ou não. O desenvolvedor também deve incluir informações de contato e canais de suporte, como e-mail, fórum de discussão e indicar ferramentas que podem ser usadas para que os usuários solicitem ajuda adicional.

**Exemplo**. O software `flosssearch` disponibiliza para os usuários informações sobre funcionalidades, tecnologias e licença utilizada, em um arquivo README, sob controle de versão, que pode ser atualizado regularmente. Porém, não há informações de contato ou sobre canais de comunicação. A Figura 3.7 mostra uma parte da excelente documentação para usuários disponibilizada no website da ferramenta `Analizo`[34].

### 3.6.3. Qualidade

O processo de garantia da qualidade do *software de pesquisa* é tão necessário quanto o código que foi escrito. *Confiabilidade* é um dos atributos de qualidade do software, em geral definido como a probabilidade do mesmo funcionar sem a ocorrência de falhas em um período específico. Testes e revisão de código são duas técnicas populares que podem ser usadas no desenvolvimento de *software de pesquisa* confiável.

#### P8. Teste de software

*Teste de software* é a atividade de executar um produto de software para verificar se ele está em conformidade com suas especificações. Testes fornecem evidências sobre a confiabilidade do software, além de promover agilidade e, possivelmente, redução nos custos de desenvolvimento, depuração e manutenção do software [Delamaro et al. 2013].

[34] `https://www.analizo.org`

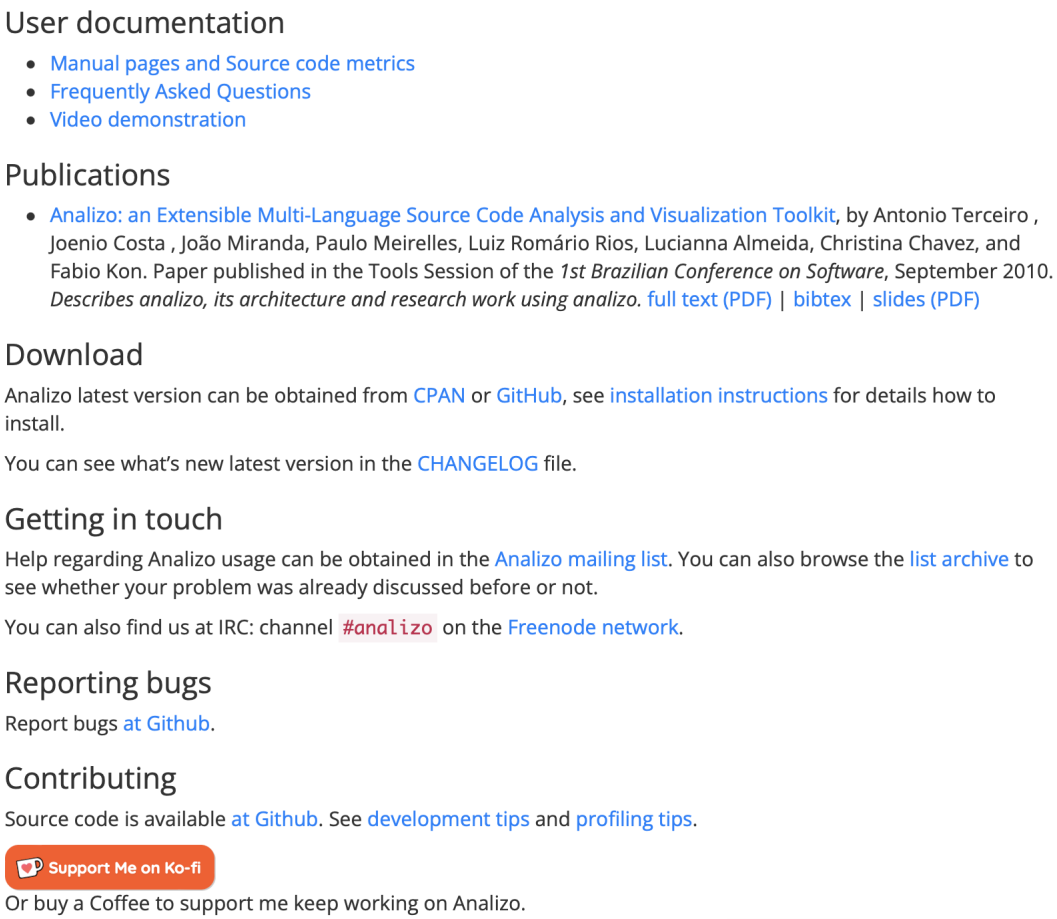



**Figura 3.7. Documentação para usuários da ferramenta `Analizo`**

O uso contínuo desta prática protege o *software de pesquisa* em relação à introdução de *bugs* em correções ou modificações futuras. *Testes de unidade* são usados para testar métodos, funções, *scripts* e outros tipos de unidades de programa, concentrando-se nas saídas geradas em resposta às entradas e às condições de execução [Delamaro et al. 2013]. Em geral, os *testes funcionais* tratam o sistema como uma "caixa preta", pelo fato de não haver acesso aos detalhes de código para a criação dos casos de teste.

Há diversos *frameworks* disponíveis para criação e *automação de testes* funcionais e de unidade para linguagens de programação popularmente usadas no desenvolvimento de *software de pesquisa*, por exemplo, *JUnit* (http://junit.org/) para Java, *CUnit* (http://cunit.sourceforge.net/) para C, *py.test* (http://pytest.org/) para Python e *PHPUnit* (https://phpunit.de) para PHP.

**Exemplo.** A ferramenta de análise estática `Analizo` possui um conjunto de testes automatizados. A Figura 3.8 mostra informações sobre como executar seus testes.

Apesar da linguagem PHP possuir um framework para automação de testes, o *PHPUnit*, o software `flosssearch` não possui testes automatizados. A falta de testes pode desestimular os desenvolvedores para consertar, estender ou melhorar o *software de pesquisa*, e desencorajar outros pesquisadores a usá-lo.

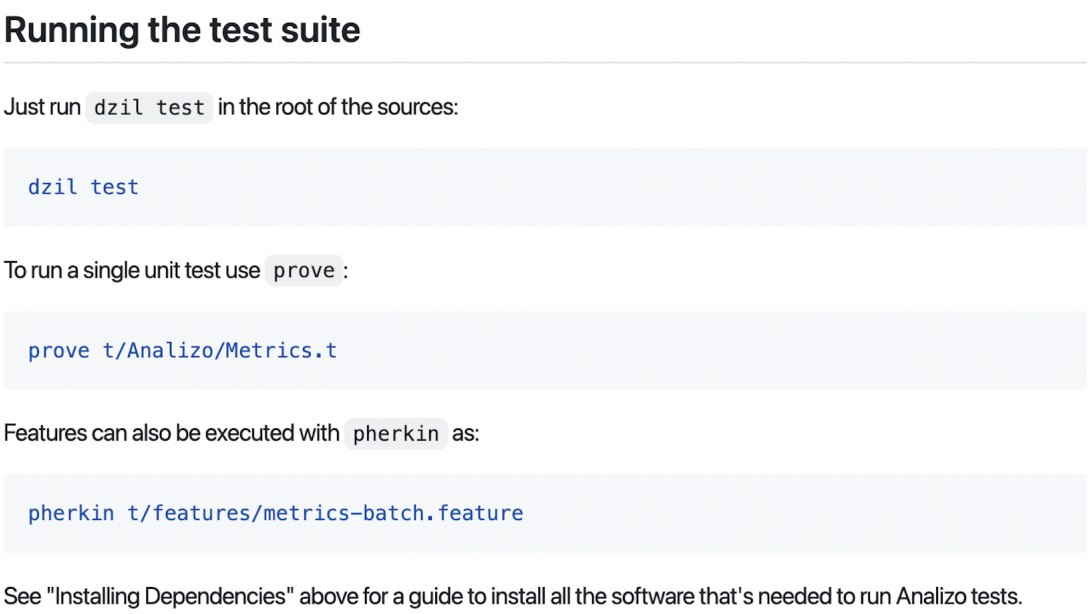



**Figura 3.8. Trecho da documentação no GitHub mostrando como executar os testes da ferramenta `Analizo`.**

### P9. Revisão de código

Revisão de código, ou revisão por pares, é uma prática consolidada para a melhoria da qualidade do software. A colaboração entre pares durante a revisão é facilitada por meio do acesso compartilhado ao código-fonte e vários canais de comunicação. Outros desenvolvedores podem perceber problemas na qualidade do código do *software de pesquisa* que testes automatizados podem não detectar. Os revisores podem identificar alguns tipos de *bugs*, problemas lógicos, falta de padronização e de conformidade com as práticas e diretrizes do projeto.

No contexto do *software de pesquisa*, a revisão de código pode, além de promover a melhoria da qualidade do software, ser aplicada a outros ativos da pesquisa. A revisão de código também permite que outros desenvolvedores e pesquisadores avaliem se os métodos aplicados e implementados no *software de pesquisa* estão em conformidade com as teorias subjacentes da pesquisa. A falta de conformidade entre teoria e implementação em um *software de pesquisa* pode levar a conclusões errôneas em todo um estudo e seus resultados.

**Exemplo.** Ao ser colocado em um repositório público, o software `flosssearch` foi compartilhado e pôde ser revisado por outros pesquisadores. O texto de apresentação mostrado na tela principal do `flosssearch` em execução, foi revisado por outro pesquisador do grupo e algumas inconsistências foram reportadas e corrigidas.

## 3.6.4. Gerência

As práticas de gerência apresentadas estão relacionadas aos processos e ferramentas usados para acompanhar e aumentar a eficiência de tarefas de desenvolvimento do *software de pesquisa*.

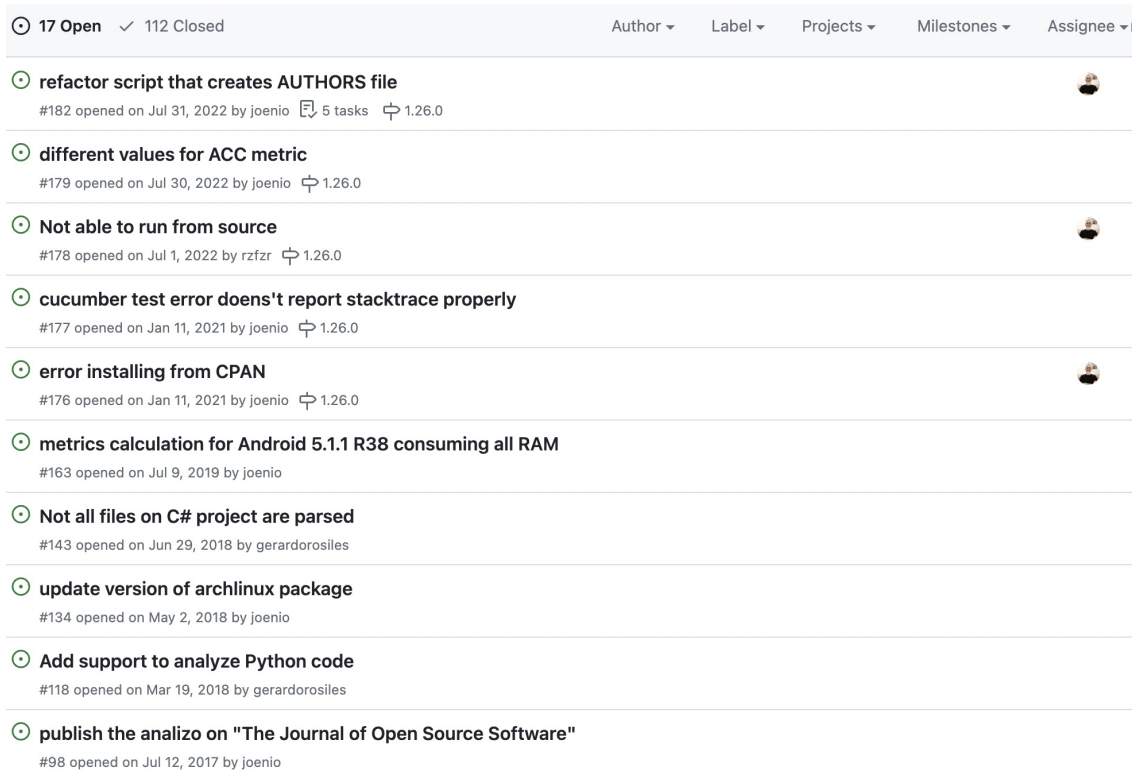



**Figura 3.9. *Issue tracker* da ferramenta `Analizo` no GitHub.**

### P10. *Issue Trackers*

*Issue trackers*[35] são ferramentas para rastrear e gerenciar problemas, em geral associados a tarefas como resolução de *bugs* e solicitações de melhoria para um software. *Issue trackers* facilitam a colaboração e permitem que os colaboradores acompanhem o progresso das atividades e trabalhem em conjunto para resolvê-las. Para indicar que o trabalho está em andamento, um desenvolvedor pode vincular uma tarefa (*issue*) ao envio de uma solicitação de melhoria no projeto (*pull request*).

Em geral, plataformas de hospedagem de repositórios, como Github, Gitlab[36], Codeberg[37], e Launchpad[38] oferecem um *issue tracker* nativo e integrado. Alternativas estão disponíveis em ferramentas externas, específicas para gestão de tarefas, dentre elas, Jira, Bugzilla, Trello, e Trac.

No contexto de um *software de pesquisa*, o *issue tracker* também pode ser utilizado para acompanhar a escrita colaborativa de artigos científicos em formato texto simples, usando linguagens de marcação (*Markdown* ou para *LaTeX*), para posterior processamento.

**Exemplo.** A Figura 3.9 apresenta a página do *issue tracker* da ferramenta *Analizo* no

[35] Usamos o termo em inglês, *issue tracker*, ao invés de traduzí-lo para *rastreador de problemas*, dado que o primeiro é amplamente utilizado pela comunidade internacional de desenvolvedores de software.

[36] `https://gitlab.com`

[37] `https://codeberg.org`

[38] `https://launchpad.net`

GitHub. O projeto recebe solicitações de pessoas externas ao projeto e seus desenvolvedores interagem com os *tickets* criados na tentativa de entender o problema ou ajudar na solução. O software `flosssearch` não usa o recurso de *issue tracker* do GitHub, nem ferramentas externas com essa finalidade.

### P11. Automatização de tarefas

Durante o desenvolvimento de software muitas tarefas são realizadas de forma repetitiva. A automatização permite que essas tarefas sejam executadas de forma rápida e consistente por meio de *scripts*, ferramentas ou sistemas automatizados. Além dos testes, outras tarefas podem ser automatizadas para ajudar a encontrar e investigar *bugs* mais rapidamente, melhorar a qualidade do software e reduzir o tempo para validação e lançamento de versões do software. A automatização pode ser configurada para executar diferentes tipos de testes (unitários, integração), para analisar o código-fonte em busca de problemas de estilo, convenções de codificação ou más práticas, entre outras tarefas.

**Exemplo.** A Figura 3.10 mostra um exemplo de automatização de tarefas e apresenta um arquivo com a definição de passos para a execução dos testes unitários do *software de pesquisa* `Parcels`.

### P12. Integração e implantação contínuas

*Integração Contínua* ("Continuous Integration") e *Implantação Contínua* ("Continuous Deployment") são práticas de automação no desenvolvimento de software que possibilitam a entrega rápida e confiável de um software. Na integração contínua, as mudanças são automaticamente verificadas e integradas, permitindo que os problemas sejam detectados e corrigidos mais cedo no processo de desenvolvimento. A implantação contínua permite que as mudanças sejam disponibilizadas automaticamente para os usuários.

Para configurar um sistema de integração contínua é necessária a configuração de uma ferramenta, como *Jenkins*, *CircleCI*, *Travis CI* ou *GitHub Actions*, para monitorar o repositório do software e executar automaticamente testes e tarefas de verificação. A Figura 3.11 mostra um *pull request* no repositório do *software de pesquisa* `Parcels`. O projeto implementa integração contínua e, a cada *pull request* nesse repositório, tarefas automatizadas são executadas, como testes unitários e de integração e uso de uma ferramenta de análise de código-fonte (*linter*) para identificar possíveis problemas no código. Como as tarefas são obrigatórias, o código não poderá ser incorporado ao *branch* principal se alguma das atividades falhar. Na Figura 3.11, podemos observar que os testes unitários falharam.

Os *pipelines* de implantação contínua são fluxos de trabalho automatizados que ajudam a *implantar novas versões do software em diferentes ambientes*. Eles podem ser criados usando ferramentas como *Docker*, *Kubernetes*, *GitLab CI/CD* e *CircleCI*. Para exemplificar a prática, a Figura 3.12 mostra as tarefas automatizadas realizadas quando um *pull request* é incorporado ao projeto na *branch* principal. O projeto é o *software de*

```yaml
name: unit-tests
on:
  push:
    branches:
      - "master"
      - "test-me/*"
  pull_request:
    branches:
      - "*"
  schedule:
    - cron: "0 7 * * 1" # Run every Monday at 7:00 UTC

defaults:
  run:
    shell: bash -el {0}

jobs:
  unit-test:
    name: Unittesting on ${{ matrix.os }} with python latest
    runs-on: ${{ matrix.os }}-latest
    strategy:
      fail-fast: false
      matrix:
        os: [macos, ubuntu, windows]
        include:
          - os: macos
            os-short: osx
          - os: ubuntu
            os-short: linux
          - os: windows
            os-short: win
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Setup Conda and parcels
        uses: ./.github/actions/install-parcels
        with:
          environment-file: environment_py3_${{ matrix.os-short }}.yml
          environment-name: py3_parcels
      - name: Unit test
        run: |
          coverage run -m pytest -v -s --html=${{ matrix.os }}_unit_test_report.html --self-contained-html tests
          coverage xml
      - name: Codecov
        uses: codecov/codecov-action@v3.1.1
        with:
          flags: unit-tests
      - name: Upload test results
        if: ${{ always() }} # Always run this step, even if tests fail
        uses: actions/upload-artifact@v3.1.2
        with:
          name: Unittest report
          path: ${{ matrix.os }}_unit_test_report.html
```

**Figura 3.10. Fluxo de trabalho automatizados no repositório do *software de pesquisa* `Parcels`**

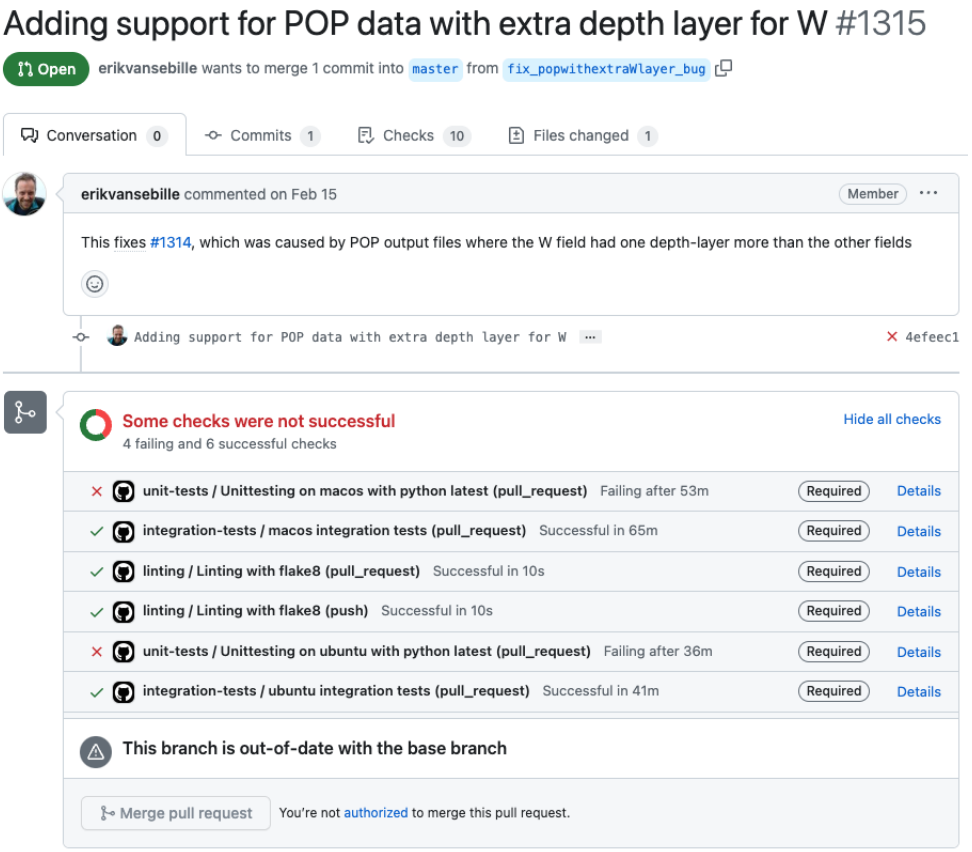



**Figura 3.11. *Pull request* com testes unitários falhando no repositório do projeto Parcels**

*pesquisa* GGIR[39], que converte dados brutos de dispositivos vestíveis em relatórios para pesquisadores que investigam a atividade física diária e o sono humano. Após a aprovação do *pull request*, as tarefas automatizadas são executadas e o lançamento da versão.

Ao adotar essas práticas de integração e implantação contínuas no *software de pesquisa*, a confiabilidade do software será maior e a entrega de novas funcionalidades será mais rápida e segura.

### P13. Lançamento de versões

O *lançamento de versões* é importante durante o desenvolvimento de *software de pesquisa* sustentável para permitir a rastreabilidade das mudanças e permitir que a pesquisa seja reproduzida a partir de uma versão específica do software. O lançamento de versões envolve a marcação de pontos específicos no histórico do código-fonte para representar marcos no desenvolvimento do software. Os repositórios de código fonte fornecem vários recursos que apoiam o lançamento de versões.

Uma estratégia comum é o uso de *tags* (etiquetas), ou marcadores atribuídos a um *commit* para identificar uma versão específica do software. As *tags* podem ser usadas para

[39] `https://github.com/wadpac/GGIR`


**Figura 3.12. Tarefas automatizadas executadas após a incorporação de um *pull request* no repositório do *software de pesquisa*.**

identificar versões estáveis, versões de correção de *bugs*, pontos de lançamento importantes ou algum *milestone* (marco) no desenvolvimento do projeto. Por exemplo, uma *tag* pode ser criada para indicar a primeira versão funcional do software ("versão 1.0").

Outro recurso importante é o uso de *releases* (lançamentos), com software empacotado e disponibilizado para os usuários. *Pacote* (*package*) é o termo utilizado para representar um conjunto organizado de arquivos e recursos relacionados que são distribuídos juntos como uma unidade coesa. O pacote pode incluir o código-fonte do software, dependências, arquivos de configuração, documentação e qualquer outro componente necessário para executar ou instalar o software. A disponibilização do *software de pesquisa* como pacote facilita a distribuição, instalação e utilização do software, entregando um conjunto funcional de recursos e funcionalidades para os usuários. Cada *release* pode incluir *release notes* (notas de lançamento) detalhando as alterações, correções de *bugs* e outras informações relevantes.

O nome da versão geralmente segue uma convenção numérica ou alfanumérica, como "1.0", "2.3" ou também pode incluir um identificador descritivo, como "versão beta" ou "versão alpha". O versionamento semântico[40] é uma prática comum fortemente recomendável ao adotar e usar versionamento para gerenciar as *releases* do *software de pesquisa*. O *escopo de uma versão* engloba as alterações e atualizações contidas na versão específica, inseridas pelos *commits* incluídos desde a versão imediatamente anterior até a versão sendo lançada. Recomenda-se incluir na descrição da *release* informações sobre o seu escopo, por exemplo, funcionalidades adicionadas, problemas resolvidos, melhorias de desempenho, atualizações de segurança e outros detalhes relevantes. Essas informações ajudam a comunicar claramente o que uma determinada versão oferece aos usuários.

**Exemplo.** O *software de pesquisa* `Parcels` está em um repositório público e utiliza as funcionalidades de *tags* que descrevem o escopo da versão nos lançamentos de versões. A Figura 3.13 apresenta a lista de *tags* no repositório e a Figura 3.14 mostra as notas do lançamento de uma versão do projeto.

[40] `https://semver.org/lang/pt-BR`

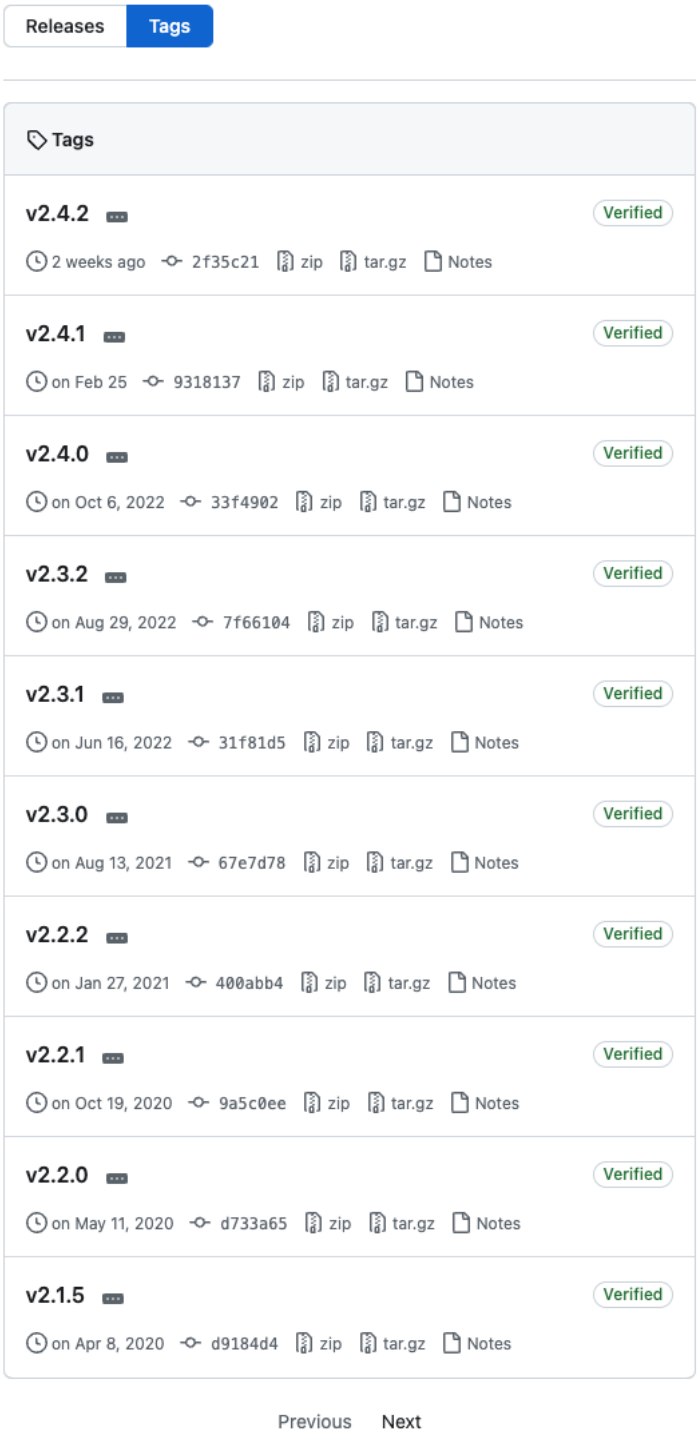



**Figura 3.13. Lista de *tags* no repositório do software de pesquisa `Parcels`.**

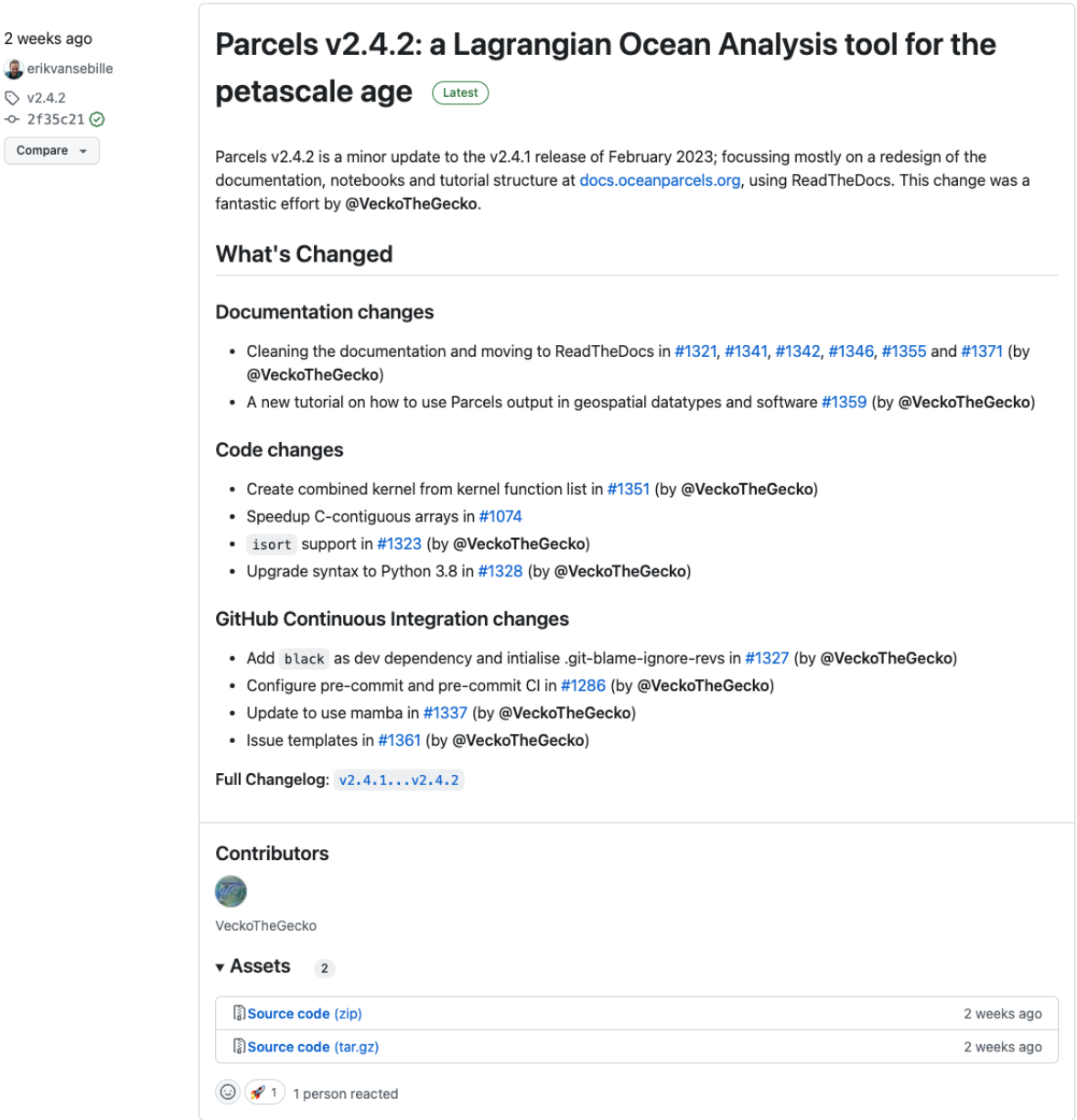



**Figura 3.14. Notas de lançamento de uma versão no repositório do software de pesquisa Parcels.**

### 3.6.5. Reconhecimento

As boas práticas que promovem a sustentabilidade técnica do *software de pesquisa* podem atrair usuários e contribuidores para o software. As práticas apresentadas a seguir estão relacionadas com o reconhecimento da relevância do *software de pesquisa* para outros pesquisadores e da necessidade de sua divulgação e citação apropriadas.

#### P14. Comunidade do projeto

Em geral, o *software de pesquisa* é desenvolvido por indivíduos ou em pequenos grupos de pesquisa dentro de uma única instituição, resultando em protótipos para atender às suas próprias necessidades, por exemplo, validar uma hipótese ou apoiar atividades da pesquisa. Criar uma comunidade em torno de um *software de pesquisa* é essencial para aumentar a adoção e colaboração, ampliando seu impacto na comunidade científica. Além disso, a comunidade ajuda na identificação e correção de problemas e traz novas sugestões para o desenvolvimento do software.

Uma comunidade ativa e engajada promove a colaboração entre pesquisadores e desenvolvedores de diferentes instituições e áreas de pesquisa. Tal engajamento facilita a troca de ideias, a discussão de problemas, a identificação de *bugs*, a proposição de novas funcionalidades, a implementação de melhorias e novas funcionalidades para o software. Outra vantagem da comunidade é realizar revisão por pares do código-fonte, documentação e resultados produzidos pelo software. Essas revisões ajudam a garantir a qualidade e a confiabilidade do software. A diversidade de perspectivas e experiências na comunidade contribui para um processo de desenvolvimento mais completo. Além disso, a existência de uma comunidade ativa em torno do *software de pesquisa* pode aumentar sua sustentabilidade no longo prazo. Com uma base de usuários e desenvolvedores engajados, o software tem maior probabilidade de receber suporte financeiro, recursos e atualizações contínuas, facilitando sua disponibilidade e relevância para outros pesquisadores.

Para envolver usuários e colaboradores em um *software de pesquisa*, é importante estabelecer diretrizes claras, reconhecer e valorizar as contribuições, disponibilizar canais de comunicação, organizar eventos e atividades, fornecer documentação detalhada e acessível e incentivar a colaboração ativa.

As diretrizes que orientam a participação na comunidade abordam aspectos sobre como contribuir com código, como reportar problemas e participar em discussões. Idealmente, o projeto deve disponibilizar um código de conduta e um guia sobre como contribuir. Quando definidas e seguidas, diretrizes ajudam a criar um ambiente colaborativo e respeitoso. Outra estratégia essencial é reconhecer e valorizar as contribuições dos usuários e colaboradores. Com agradecimentos públicos, inclusão de nomes nos créditos do projeto, menções em publicações relacionadas ao *software de pesquisa* e outros gestos de reconhecimento, incentivam-se o engajamento contínuo e a motivação para continuar colaborando com o projeto.

Para facilitar a interação entre os membros da comunidade é fundamental disponibilizar *canais de comunicação*, como fóruns de discussão online, listas de e-mails, salas de bate-papo e plataformas como GitHub e GitLab. Esses canais permitem que

os colaboradores obtenham suporte técnico, compartilhem ideias, troquem experiências e contribuam com melhorias no *software de pesquisa*. Também é possível promover a interação direta entre os membros da comunidade com a organização ou participação em conferências, workshops, treinamentos e eventos.

Para incentivar a colaboração ativa dos usuários é importante a identificação de tarefas específicas que precisam de contribuições, a criação de programas de mentoria para novos colaboradores, orientações claras sobre como contribuir para o projeto. Por fim, é essencial fornecer uma documentação detalhada e acessível para que os colaboradores possam compreender e utilizar o software de forma eficaz.

**Exemplo.** A ferramenta `flosssearch` foi implementada em PHP, por conveniência e experiência do desenvolvedor, pensando na construção de uma comunidade que, no futuro, poderia transformar `flosssearch` em um produto. Porém não há políticas ou recomendações definidas para os membros da comunidade.

A plataforma GitHub apresenta suas políticas e recomendações para os membros da comunidade[41], incentivando-os a moderar seus projetos sempre que possível e denunciar qualquer conteúdo que possa violar tais políticas.

### P15. Divulgação de Software

A divulgação do *software de pesquisa* em eventos científicos e estudos com outros pesquisadores é de extrema importância. Essas oportunidades permitem que os pesquisadores compartilhem e apresentem suas soluções de software, que avaliem a usabilidade da ferramenta e potenciais interesses em comum.

Ao apresentar o *software de pesquisa* para a comunidade acadêmica e colaborar com outros pesquisadores, é possível receber opiniões, estabelecer parcerias, trocar conhecimento e, talvez, aumentar o impacto e a adoção do software de pesquisa, impulsionando o alcance da ciência.

**Exemplo.** O software `flosssearch` foi utilizado e publicado em artigos científicos, tanto como instrumento da pesquisa, como produto da pesquisa. Entretanto, ainda não há informações documentadas no repositório sobre a correspondência entre artigos publicados e versões do software.

O arquivo README do `Parcels` apresenta uma seção sobre "manuscrito e código", onde cita artigos publicados e a versão software usada:

> The manuscript detailing the performance of `Parcels v2.4` is available at Computers & Geosciences and can be cited as:
>
> Kehl, C, PD Nooteboom, MLA Kaandorp and E van Sebille (2023). *Efficiently simulating Lagrangian particles in large-scale ocean flows – Data structures and their impact on geophysical applications*, Computers and Geosciences, 175, 105322. https://doi.org/10.1016/j.cageo.2023.105322

[41] `https://docs.github.com/en/site-policy/github-terms/github-community-guidelines`

### P16. Citação de Software

A boa prática científica exige que o *software de pesquisa* mencionado em publicações científicas seja mantido para reprodutibilidade e verificação de resultados científicos. Citar um software de pesquisa é uma forma de reconhecer e dar visibilidade aos autores e ao próprio software, possibilitando que outros pesquisadores o localizem e eventualmente reusem. Entretanto, a citação de software em publicações científicas ainda não é uma prática comum entre cientistas, mas padrões de citação têm sido propostos e adotados.

Em seu artigo "Recognizing the value of software: a software citation guide", [Katz et al. 2020] apresentam instruções simples sobre como o software pode se tornar citável e destacam que o uso de identificadores persistentes (PIDs) e metadados descritivos são elementos essenciais da citação de software. Informações sobre autoria, nome, local de publicação, data de publicação e identificador do software são consideradas obrigatórias. Adicionalmente, alguns estilos de citação pedem uma descrição da citação entre colchetes. Para software, pode-se usar *[Computer software]*. A Tabela 3.3 apresenta o formato de citação de software proposto por [Katz et al. 2020].

**Tabela 3.3. Formato de Citação de Software [Katz et al. 2020].**

| Informação | Descrição |
|---|---|
| **Criador(es)** | Autor(es) ou projeto que desenvolveram o software |
| **Título** | Nome do software |
| **Local** | Local de publicação do software, por exemplo, um arquivo ou repositório que fornece identificadores persistentes. |
| **Data** | Data em que o software foi publicado, por exemplo, a data associada a um lançamento ou versão do software. |
| **Identificador** | Um apontador resolvível para o software, de preferência um PID que leve para uma página contendo metadados descritivos sobre o software, semelhante a um DOI. |

O formato *Citation File Format* (CFF)[42] tem se popularizado como formato e padrão para documentar como citar um projeto de software [Druskat 2021]. Adicionar um arquivo *CITATION.cff* ao repositório do *software de pesquisa* pode ser uma boa forma documentar e atualizar as recomendações definidas para citação do software. Recentemente, Zenodo e GitHub facilitaram a adição de metadados aos repositórios e a geração de citações para esses repositórios [Smith 2021].

Além disso, pode-se definir um ID persistente para o *software de pesquisa*, por exemplo, um DOI obtido para um artigo que apresenta o software para a comunidade (*software paper*) ou ainda usar formatos mais recentes como o SWHID[43] promovido pelo projeto Software Heritage[44], uma iniciativa dedicada a coletar e preservar software em formato de código-fonte para preservação e acesso público. Entretanto, [Katz et al. 2020] recomendam que, se existir um artigo que apresenta o *software de pesquisa* para a comu-

[42] `https://citation-file-format.github.io`
[43] `https://www.swhid.org`
[44] `https://www.softwareheritage.org`

nidade científica, ele deve ser citado como uma referência bibliográfica *adicional*, mas não deve substituir a citação do software propriamente dita.

**Exemplo.** O *software de pesquisa* `flosssearch` não definiu como deve ser citado. A ferramenta `Analizo` indica uma publicação que pode ser considerada como citação para o software [Terceiro et al. 2010] mas não apresenta uma citação de software.

## 3.7. Avaliação do *software de pesquisa* `MoSyn`

Nesta seção, apresentamos os resultados de um estudo exploratório conduzido com um grupo de pesquisa da área de Física com o objetivo de identificar problemas relacionados ao *software de pesquisa* desenvolvido pelo grupo, fazer uma caracterização inicial da sustentabilidade e aderência aos princípios FAIR de um *software de pesquisa* e recomendar boas práticas para torná-lo mais sustentável e aberto.

### 3.7.1. Entrevista com Pesquisador

A primeira atividade do estudo foi a realização de uma entrevista com um pesquisador sênior e líder de grupo de pesquisa da área de Física Aplicada, com mais de 30 anos de experiência. Seu interesse no desenvolvimento de *software de pesquisa* é motivado pelo desejo de utilizar a computação como ferramenta para avançar o conhecimento em sua área de pesquisa. As respostas forneceram importantes considerações sobre o uso de práticas de sustentabilidade no *software de pesquisa* desenvolvido por seu grupo.

De forma geral, as opiniões do pesquisador endossam fortemente o uso de *software de pesquisa* sustentável para o grupo de pesquisa. Ele reconhece o valor desse trabalho e o desejo de contribuir para a comunidade de pesquisa em geral. Sua crença nos benefícios da sustentabilidade, aliada ao entusiasmo em tornar o software acessível e ao reconhecimento da importância da replicação, reforça ainda mais o apoio do pesquisador às práticas de *software de pesquisa* sustentável. Ele destaca a necessidade de mais conhecimento por parte dos desenvolvedores do grupo de pesquisa. Os integrantes do grupo valorizariam ter suporte para entender e implementar essas práticas de forma eficaz.

Os resultados também destacam as considerações e dilemas relacionados ao desejo de abertura, à proteção de suas contribuições intelectuais e ao receio de comprometer a integridade e a credibilidade de sua pesquisa ao compartilhá-la prematuramente. A falta de familiaridade dos desenvolvedores de *software de pesquisa* com as melhores práticas de desenvolvimento de código também é considerada uma barreira. O grupo de pesquisa inclui pesquisadores com formação acadêmica em diferentes áreas, sendo que sua *expertise* principal está nas respectivas áreas de pesquisa, e não na engenharia de software.

Durante a entrevista, o pesquisador inicialmente tinha pouco conhecimento sobre as práticas de sustentabilidade em *software de pesquisa* , mas demonstrou entusiasmo e aceitou as práticas apresentadas assim que foram introduzidas, reconhecendo os benefícios e a importância da adoção de práticas de sustentabilidade no desenvolvimento de *software de pesquisa*. Outro fator de apoio identificado é o compromisso com a promoção da reprodutibilidade em sua pesquisa. Com os dados e o código publicamente disponíveis, a pesquisa pode ser replicada e potencialmente expor quaisquer erros ou inconsistências, contribuindo para a confiabilidade e robustez do conhecimento científico como um todo.

A atitude positiva do pesquisador em relação ao *software de pesquisa* sustentável reflete uma apreciação genuína pelos benefícios que ele oferece ao seu trabalho. O pesquisador deseja que seu software esteja disponível para todos, reconhecendo seu potencial para auxiliar inúmeros pesquisadores. Ele enfatiza as aplicações práticas dos resultados da pesquisa em ambientes clínicos, onde o software poderia ser usado para avaliar e tratar pessoas de forma eficaz. Além disso, o pesquisador destaca a importância da replicação na pesquisa e o papel do software sustentável em garantir resultados precisos e confiáveis, reconhecendo que erros e *bugs* são inerentes ao software. Ele e seu grupo de pesquisa estão interessados em identificar e corrigir esses problemas. O pesquisador também reconhece os benefícios dos testes automatizados na garantia de que as novas versões do software sejam confiáveis e na correção de *bugs* anteriores, economizando assim tempo e esforço valiosos. O pesquisador compreende os benefícios de tornar o *software de pesquisa* sustentável e de colaborar para avançar o conhecimento científico.

No final da entrevista, solicitamos que o pesquisador indicasse alguns projetos de *software de pesquisa* para avaliação de sustentabilidade. Os resultados da avaliação, reportados em um relatório técnico, com sugestão de práticas e melhorias, seriam enviados para o pesquisador. Para a primeira avaliação de sustentabilidade, escolhemos o software `MoSyn`[45], um aplicativo para análise de grafos variáveis no tempo.

### 3.7.2. Avaliação da Sustentabilidade

O *software de pesquisa* `MoSyn` é um aplicativo baseado em MATLAB, projetado para a análise de grafos variantes no tempo (TVGs) e suas medidas associadas. A ferramenta fornece uma estrutura modular com várias classes e funções para lidar com diferentes aspectos da análise, como configuração, recursos gráficos e gerenciamento de projetos.

A Tabela 3.4 apresenta um resumo da avaliação da sustentabilidade do software `MoSyn`. A seguir, apresentamos trechos do *Relatório de Avaliação* do software `MoSyn` com referências às práticas apresentadas na Tabela.

**# Relatório de Avaliação da Sustentabilidade do Software `MoSyn`**

**## Práticas Básicas**

O software `MoSyn` esteve hospedado em um repositório público desde o início de seu desenvolvimento (**P1**). Apesar da hospedagem do *software de pesquisa* no GitHub e uso das facilidades da plataforma, um backup periódico tem sido realizado em um dispositivo de armazenamento do grupo de pesquisa, mantido nas instalações do grupo. No GitHub a identidade do software é clara e única: o nome público do software é `MoSyn`. Apesar de ter um arquivo README.md, não há uma descrição do projeto que facilite sua indexação nos mecanismos de busca.

Por estar hospedado no GitHub, `MoSyn` possui suporte para controle de versão (**P2**). Inicialmente, o repositório do projeto não apresentava a licença de software (**P3**). Após quatro meses da realização da entrevista com o pesquisador sênior do grupo, houve um *commit* no repositório do software para a inclusão do arquivo *LICENSE*, descrevendo a licença escolhida, porém sem deixar claro se as permissões se aplicam a qualquer ar-

[45] `https://github.com/mpnetto/MoSyn`

**Tabela 3.4. Sustentabilidade do software `MoSyn`.**

| P | Descrição | Atende? | Comentário |
|---|---|---|---|
| P1 | O projeto está hospedado em um repositório público | Sim | O software está disponibilizado em um repositório público no GitHub |
| P2 | O software implementa controle de versão | Sim | O controle de verão é implementado por utilizar o GitHub como plataforma de hospedagem |
| P3 | Uma licença de software foi adotada | Parcialmente | Um arquivo declarando a licença MIT. Porém não está claro se as permissões se aplicam a qualquer arquivo fonte no repositório |
| P4 | O software está publicado formalmente e apresentam um DOI | Não | O repositório não menciona um DOI associado a ele mas pode ser encontrado no GitHub pelo nome |
| P5 | A estrutura de arquivos comunica a finalidade dos elementos do projeto | Sim | A estrutura de pastas está organizada de forma descritiva e permite inferir o conteúdo |
| P6 | Adota formatos de dados e interfaces comuns | Sim | Apesar de não haver documentação explícita, o software utiliza o formato de entrada e saída que facilita a integração com o MATLAB |
| P7 | A documentação apresenta uma visão geral sobre o software | Parcialmente | O projeto utiliza o *GitHub pages* mas as informações estão incompletas e algumas URLs direcionam para um destino não válido. |
| P8 | O software implementa testes | Não | Não há testes automatizados para o software |
| P9 | O código é revisado antes de ser incorporado ao código | Parcialmente | Todos os *pull requests* listados no repositórios foram aprovados pelo próprio autor, sugerindo que não houve revisão de código por outra pessoa |
| P10 | Disponibiliza e usa *issue tracker* | Sim | O projeto aproveita a funcionalidade de rastreamento de *bugs* e tarefas disponível no GitHub |
| P11 | As tarefas repetitivas são automatizadas | Não | Não encontramos tarefas automatizadas no projeto |
| P12 | Há integração e implantação contínua | Não | Não há integração contínua. Por ser um plugin instalado manualmente no MATLAB, a implantação contínua não é viável |
| P13 | O software faz lançamento de versões | Não | O projeto não utiliza a funcionalidade de lançamento de versões no repositório do GitHub |
| P14 | Há evidência de uma comunidade (presente ou futuro) | Não | Há apenas um desenvolvedor como autor |
| P15 | O software é divulgado em eventos científicos | Não | Não encontramos divulgação em eventos científicos |
| P16 | O software é citado em publicações científicas | Não | Não encontramos citação do software em publicações. |

quivo fonte do repositório. A licença atribuída ao software `MoSyn` é a MIT[46], um licença usada em projetos de software livre e em projeto de software proprietário. Quanto ao registro do software (**P4**), o repositório não menciona se há um DOI associado a ele. Além disso, não encontramos uma referência para o software `MoSyn` no Zenodo.

**## Organização do Projeto**

O repositório do `MoSyn` possui uma estrutura de arquivos (**P5**) bem definida, e usa nomes auto-explicativos para pastas e arquivos que facilitam a compreensão do propósito e utilidade do *software de pesquisa*. O software `MoSyn` também se preocupa com padronização (**P6**) visto que é uma aplicação que depende do software MATLAB[47], uma plataforma paga para programação e computação numérica usada por engenheiros e cientistas para analisar dados, desenvolver algoritmos e criar modelos. Assim, o `MoSyn` utiliza formatos de entrada e saída que facilitam a integração com o MATLAB.

No `MoSyn`, a preocupação com documentação do software (**P7**) ainda é incipiente e voltada para usuários do software. Inicialmente, o repositório não tinha qualquer documentação. Quatro meses após a entrevista realizada com o pesquisador, um arquivo *README* foi adicionado ao repositório com descrição da ferramenta, lista de funcionalidades, informações sobre utilização e contribuição e licença utilizada pelo software (**P7**). Os autores do software não estão listados em um arquivo, mas é possível identificar as pessoas que contribuíram com o software a partir de informações sobre autores das mudanças registradas pelo sistema de controle de versão (*commits*). Finalmente, os desenvolvedores deram início à construção de um site para publicação de informações sobre o projeto de pesquisa e o software usando o *GitHub Pages*.

**## Qualidade**

O *MATLAB Test*[48] é um conjunto de ferramentas para o desenvolvimento, gerenciamento, análise e teste de aplicações MATLAB. Entretanto, o *software de pesquisa* `MoSyn` ainda não implementa testes de software automatizados (**P8**). Vale destacar que as ferramentas do *MATLAB Test*, assim como o MATLAB, são produtos de software fechados e que requerem assinaturas pagas.

Considerando que o software `MoSyn` está publicado no GitHub, é possível realizar a revisão de código (**P9**) colaborativamente, utilizando a infraestrutura oferecida pela plataforma. Porém, até o dia da avaliação do *software de pesquisa*, todos os pedidos de mudança no código (*pull requests*) listados no repositório foram aprovados pelo próprio autor, sugerindo que não houve revisão de código por outra pessoa.

**## Gerência**

O software `MoSyn` está hospedado no GitHub e conta com o seu rastreador de tarefas e *bugs* nativo (**P10**) nativo. O *GitHub Docs*[49], apresenta uma breve introdução sobre como reportar um problema usando o rastreador de tarefas. Entretanto, o rastreador nativo do GitHub ainda não foi utilizado no projeto `MoSyn`. Não foram encontradas tare-

[46] `https://opensource.org/license/mit/`
[47] `https://www.mathworks.com/products/matlab.html`
[48] `https://www.mathworks.com/products/matlab-test.html`
[49] `https://docs.github.com/pt`

fas automatizadas ou indício de automatização de tarefas (**P11**). Por fim, não há suporte para integração e implantação contínuas (**P12**). Por ser um *plugin* instalado manualmente pelo usuário no MATLAB, tais práticas não são viáveis.

O projeto não segue a prática de lançamento de versões (**P13**) no repositório do GitHub. As funcionalidades de 'Releases' e 'Tags' que facilitariam a referência e a descrição das funcionalidades e bugs incluídos naquela versão específicas não são utilizadas. Apesar de não realizarem lançamentos oficiais de versões, se fosse necessário citar o *software de pesquisa* em um artigo, os autores poderiam fazer referência a um *commit* específico no repositório do projeto. No histórico do projeto vimos que um *commit* com a mensagem "mosyn 2.0" foi incorporado ao projeto quatro meses após a entrevista. Essa mensagem sugere uma intenção de indicar uma alteração significativa no projeto e associar um número de versão.

**## Reconhecimento**

O projeto ainda não possui uma comunidade (**P14**). Apenas o usuário dono do repositório submeteu *commits* e *pull requests* no projeto. A página do projeto no GitHub não apresenta outros usuários que adicionaram o projeto como favorito e não mostra usuários que fizeram uma cópia independente (*fork*) do projeto.

Não encontramos artigos ou outras formas de divulgação do *software de pesquisa* em eventos científicos (**P15**). O software `MoSyn` / MATLAB é mencionado e foi usado em uma dissertação de mestrado na área de Saúde para extrair índices de redes funcionais cerebrais (RFC) para análise estatística [Toutain 2019]. Entretanto, não encontramos no texto da dissertação um identificador ou referência para o software ou para a versão usada. Também não encontramos citação do software em publicações e na dissertação mencionada (**P16**), nem informações sobre como o software deveria ser citado.

### 3.7.3. Avaliação de *FAIRness*

As Tabelas 3.5, 3.6 e 3.7 apresentam uma avaliação preliminar de *FAIRness* do software `MoSyn`. Cada linha da tabela apresenta um princípio, sua descrição, indicação se o software atende ou não ao princípio, e uma justificativa se procedente. As tabelas mostram apenas as linhas em que o software não atendeu ou atendeu parcialmente ao princípio. A seguir, discutimos como *software de pesquisa* incorporou os princípios *FAIR*.

**# Relatório de Avaliação de *FAIRness* do Software `MoSyn`**

**## F: O software e seus metadados associados são facilmente encontrados tanto por humanos quanto por máquinas**

O software `MoSyn` está hospedado com uma identidade clara e única no GitHub, não globalmente e o repositório não garante a persistência do identificador se o software for movido, por exemplo (**F1**). Os componentes do software, como classes e bibliotecas, apresentam identificadores distintos (**F1.1**). Cada versão do software pode ser unicamente identificada por um *hash de commit* e, a partir dele, é possível recuperar os metadados da versão específica (**F1.2**). O software apresenta metadados, mas não descreve as dependências, informações detalhadas sobre utilização e configuração e como deve ser a entrada e saída de arquivos (**F2**). Ao analisar os metadados a partir de um commit, é possível

verificar qual versão específica do software o metadado está se referindo (**F3**). Apesar de incluir um arquivo README com informações do projeto, não há uma descrição na configuração do projeto que facilite a indexação nos mecanismos de busca (**F4**).

**## A: O software e seus metadados podem ser obtidos através de protocolos padronizados**

O software `MoSyn` pode ser obtido a partir do repositório do projeto no GitHub (**A1**). Não há restrições para baixar o código-fonte, como taxas os custos relacionados à licença (**A1.1**). É possível configurar o repositório para ser acessado apenas por pessoas autorizadas (**A1.2**). Os metadados estão descritos em arquivos no repositório, então ele não seriam acessíveis caso o repositório do software não esteja mais disponíveis (**A2**).

**## I: O software interopera com outros software trocando dados e/ou metadados através da interação via interface de programação de aplicativos (APIs), descrito por meio de padrões**

O software `MoSyn` é uma aplicação para MATLAB e, portanto, interopera seguindo seu padrão, mas a forma como a interação ocorre não está explicitamente descrito (**I1**). O repositório menciona o MATLAB como pré-requisito para executar a aplicação e inclui agradecimentos pela utilização de bibliotecas e recursos externos e recursos, mas não inclui referências qualificadas, como websites ou os nomes da bibliotecas e recursos externos utilizados (**I2**).

**## R: O software é tanto utilizável (pode ser executado) quanto reutilizável (pode ser compreendido, modificado, aprimorado ou incorporado a outros softwares)**

O software `MoSyn` não disponibiliza muitas informações descrevendo como reutilizar o software (**R1**). O software está licenciado sob a licença de código aberto MIT, que permite a reutilização, mas não deixa claro se todas as partes do software seguem a mesma licença (**R1.1**). A partir dos commits e da lista de contribuidores do projeto no GitHub é possível saber quais pessoas contribuíram com o software, mas não há informações explícitas sobre como o software foi desenvolvido ou quais foram as intenções originais (**R1.2**). Os nomes e informações sobre as bibliotecas utilizadas não são mencionadas na documentação (**R2**). A comunidade MATLAB recomenda que sejam seguidas as melhores práticas de desenvolvimento de software em relação a correção, clareza e generalização e o software, em geral, atende a esses padrões (**R3**).

## 3.8. Considerações Finais

O reconhecimento da necessidade de reprodutibilidade na pesquisa científica, bem como o advento recente da criação de comitês de avaliação de artefatos de pesquisa em conferências e periódicos levaram à recomendação para que os autores disponibilizem dados de pesquisa, *software de pesquisa*, documentação e materiais usados nas publicações científicas. Nesse contexto, o *software de pesquisa* desenvolvido deve ser sustentável para que cientistas possam entender, replicar e reproduzir pesquisas anteriores ou conduzir novas pesquisas de forma eficaz. O suporte à reprodutibilidade ainda demanda que o *software de pesquisa* seja *FAIR* – localizável, acessível, interoperável e reutilizável, para responder às necessidades de pesquisa, presentes e futuras.

**Tabela 3.5. *FAIRness* no Software MoSyn – *Findable*.**

| Princ. | Descrição | Atende? | Comentário |
|---|---|---|---|
| F: | O software e seus metadados associados são facilmente encontrados tanto por humanos quanto por máquinas. | Parcialmente | |
| F1 | O software recebe um identificador globalmente único e persistente. | Parcialmente | O software possui uma identidade única apenas no GitHub e não é garantida a persistência. |
| F1.1 | Aos componentes do software que representam diferentes níveis de granularidade são atribuídos identificadores distintos. | Sim | |
| F1.2 | As diferentes versões do software recebem identificadores distintos. | Sim | |
| F2 | O software é descrito com metadados detalhados. | Parcialmente | O software menciona alguns metadados, mas sem muitos detalhes que permitam encontrar o software facilmente |
| F3 | Os metadados contêm de forma clara e explícita o identificador do software que descrevem. | Parcialmente | Os metadados não apresentam de forma clara, mas é possível obter a partir do *commit* |
| F4 | Os metadados seguem os princípios FAIR, são pesquisáveis e indexáveis. | Parcialmente | Há informações nos arquivos do repositório, mas não há descrição na configuração do projeto para facilitar indexação |

Entretanto, grande parte dos cientistas que escrevem *software de pesquisa* ainda carecem de uma formação em boas práticas de engenharia de software, por exemplo, uso de sistemas de controle de versão, documentação adequada e testes automatizados. Neste cenário, a sustentabilidade e outros atributos de qualidade do *software de pesquisa* podem ficar comprometidos e levar a erros em conclusões científicas e falta de reprodutibilidade na pesquisa que depende do software.

Portanto, ainda que demande tempo e esforço, uma mudança na forma como o desenvolvimento e a manutenção de *software de pesquisa* são realizados é necessária, bem como a definição de um modelo de sustentabilidade de software que reconheça a importância de incorporar práticas para promover a sustentabilidade no desenvolvimento de *software de pesquisa*. Um caminho natural é investir em treinamento para cientistas de diversas áreas do conhecimento sobre conceitos, boas práticas e ferramentas para o desenvolvimento de *software de pesquisa* sustentável.

Ao adotar práticas sustentáveis, os pesquisadores podem criar oportunidades para que outros especialistas na área colaborem no desenvolvimento de seu *software de pesquisa*, forneçam *feedback* e sugestões de melhorias. Além disso, a disseminação dos resultados da pesquisa e a divulgação do software, podem ampliar o impacto além da comunidade acadêmica, beneficiando o público em geral.

**Tabela 3.6. FAIRness no Software MoSyn – *Accessible*.**

| Princ. | Descrição | Atende? | Comentário |
|---|---|---|---|
| A: | O software e seus metadados podem ser obtidos através de protocolos padronizados. | Parcialmente | |
| A1 | O software é pode ser obtido por meio de seu identificador utilizando um protocolo de comunicação padronizado. | Sim | |
| A1.1 | O protocolo é aberto, gratuito e universalmente implementável. | Sim | |
| A1.2 | O protocolo permite procedimentos de autenticação e autorização, quando necessário. | Sim | |
| A2 | Os metadados são acessíveis, mesmo quando o software não está mais disponível. | Não | Caso o software não esteja mais disponível, os metadados também ficarão inacessíveis |

A discussão sobre *software de pesquisa* sustentável é incipiente e o tema tem sido pouco explorado pela comunidade acadêmica brasileira de Computação. O Simpósio Brasileiro de Engenharia de Software (SBES 2022) foi inovador ao apresentar um documento delineando Políticas de Ciência Aberta para o evento [Flach 2022], buscando encorajar os autores a disponibilizar os artefatos usados na pesquisa – dados abertos anonimizados e código do *software de pesquisa*, e despertar o interesse na reprodutibilidade dos estudos quantitativos publicados. O tema do SBES 2022 foi *Sustentabilidade de Software*, e o título de uma das palestras convidadas, proferida pelo Prof. Daniel Katz, foi *Towards Sustainable Research Software* [Katz 2022].

### 3.8.1. Institutos e Associações

Há várias iniciativas que contemplam a questão da *Sustentabilidade do Software de Pesquisa* e o apoio a cientistas e engenheiros de software, com o objetivo de promover a adoção de boas práticas no desenvolvimento de *software de pesquisa* e torná-lo mais sustentável. Alguns países da Europa, América do Norte, Ásia e África têm inaugurado institutos nacionais para apoiar cientistas que desenvolvem *software de pesquisa* e preparar uma nova categoria de profissionais: os engenheiros de *software de pesquisa* [Jiménez et al. 2017]. Entretanto, o Brasil ainda não investiu na criação de organizações ou institutos dedicados a este tema.

Dentre os institutos e associações pioneiros, destacam-se (1) *The Carpentries*[50], (2) *Software Sustainability Institute* (SSI)[51], (3) *Research Software Engineers* (RSE)[52], (4) *The Society of Research Software Engineering*[53] e (5) *Netherlands eScience Center* [Drost et al. 2020].

[50] `https://carpentries.org`
[51] `https://www.software.ac.uk`
[52] `https://researchsoftware.org/`
[53] `https://society-rse.org`

**Tabela 3.7. FAIRness no Software MoSyn – *Interoperable, Reusable*.**

| Princ. | Descrição | Atende? | Comentário |
|---|---|---|---|
| I: | O software interopera com outros software trocando dados e/ou metadados através da interação via interface de programação de aplicativos (APIs), descrito por meio de padrões. | | |
| I1 | O software lê, escreve e troca dados de acordo com padrões da comunidade relacionados ao domínio. | Parcialmente | Não está explícito como a troca de dados ocorre com o MATLAB |
| I2 | O software inclui referências qualificadas a outros objetos. | Parcialmente | As referências se resumem a menção sobre utilização, sem citar nomes, websites ou detalhes dos outros objetos |
| R: | O software é utilizável (pode ser executado) e reutilizável (pode ser compreendido, modificado, aprimorado ou incorporado a outros softwares). | Parcialmente | |
| R1 | O software é descrito com uma variedade de atributos precisos e relevantes. | Parcialmente | Não há muitas informações sobre atributos e como reutilizar o software |
| R1.1 | É atribuída uma licença clara e acessível ao software. | Sim | *MIT License.* |
| R1.2 | O software possui informações detalhadas de procedência. | Parcialmente | Não há informações sobre como o software foi desenvolvido ou quais foram as intenções originais |
| R2 | O software inclui referências qualificadas a outros softwares. | Parcialmente | Não há informações relevantes sobre dependências. |
| R3 | O software atende aos padrões relevantes da comunidade do domínio. | Sim | |

### 3.8.2. Cursos e Treinamentos

Institutos, associações e outras organizações oferecem cursos e treinamentos gratuitos sobre diversos temas ligados ao *software de pesquisa* e sua sustentabilidade. Recomendamos que visitem os websites e consultem o material disponibilizado.

- *The Carpentries* é um instituto que oferece treinamento em habilidades computacionais e ciência de dados para cientistas em todo o mundo.
- *Software Carpentry* é um projeto voluntário dedicado a ensinar habilidades básicas de computação para pesquisadores [Munk et al. 2019, Nenadic et al. 2022].
- *Library Carpentry* é uma comunidade global que ensina habilidades de software e dados para pessoas que trabalham em funções relacionadas a bibliotecas e informações [Munk et al. 2019].

- *Netherlands eScience Center* é uma fundação independente baseada na Holanda para desenvolvimento e aplicação de software de pesquisa [Drost et al. 2020].
- *The Turing Way* oferece um manual para ciência de dados reprodutível, ética e colaborativa [Community 2022].

## Referências


[Abdo 2015] Abdo, A. H. (2015). Direções para uma academia contemporânea e aberta. In *Ciência Aberta, Questões Abertas*, pages 287–306. IBCT-Instituto Brasileiro de Informação em Ciência e Tecnologia.

[Allen et al. 2017] Allen, A. et al. (2017). Engineering Academic Software (Dagstuhl Perspectives Workshop 16252). *Dagstuhl Manifestos*, 6(1):1–20.

[Anzt et al. 2021] Anzt, H. et al. (2021). An environment for sustainable research software in germany and beyond: current state, open challenges, and call for action [version 2; peer review: 2 approved]. *F1000Research*, 9(295).

[Barker et al. 2022] Barker, M., Chue Hong, N. P., Katz, D. S., and Lamprecht, A.-L. (2022). Introducing the FAIR Principles for Research Software. *Scientific Data*, 9(622).

[Becker et al. 2015] Becker, C., Chitchyan, R., Duboc, L., Easterbrook, S., Mahaux, M., Penzenstadler, B., Rodriguez-Navas, G., Salinesi, C., Seyff, N., Venters, C., Calero, C., Kocak, S. A., and Betz, S. (2015). The Karlskrona Manifesto for Sustainability Design.

[Bezjak et al. 2018] Bezjak, S. et al. (2018). *Open Science Training Handbook, doi: 10.5281/zenodo.1212496*. Zenodo.

[Carver et al. 2022] Carver, J., Weber, N., Ram, K., Gesing, S., and Katz, D. (2022). A Survey of the state of the practice for Research Software in the United States. *PeerJ Comput. Science*.

[Carver et al. 2007] Carver, J. C., Kendall, R. P., Squires, S. E., and Post, D. E. (2007). Software development environments for scientific and engineering software: A series of case studies. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, page 550–559, USA. IEEE Computer Society.

[Chue Hong et al. 2022] Chue Hong, N. P. et al. (2022). FAIR Principles for Research Software (FAIR4RS Principles), doi: 10.15497/RDA00068.

[Community 2022] Community, T. T. W. (2022). The Turing Way: A handbook for reproducible, ethical and collaborative research.

[de Souza 2023] de Souza, A. C. M. (2023). Social sustainability approaches for a sustainable software product. *ACM SIGSOFT Softw. Eng. Notes*, 48(1):38–43.

[Delamaro et al. 2013] Delamaro, M., Jino, M., and Maldonado, J. (2013). *Introdução ao Teste de Software*. Elsevier Brasil.

[Drost et al. 2020] Drost, N. et al. (2020). *Netherlands eScience Center - Software Development Guide*. Zenodo.

[Druskat 2021] Druskat, S. (2021). Making software citation easi(er) - The Citation File Format and its integrations.

[Flach 2022] Flach, C. (2022). Políticas de Ciência Aberta do Simpósio Brasileiro de Engenharia de Software (SBES).

[Flach and Kon 2021] Flach, C. and Kon, F. (2021). Software livre: Pré-requisito para a ciência aberta (pt-br). *Computação Brasil*, 1(46):12–15.

[Forschungsgemeinschaft 2022] Forschungsgemeinschaft, D. (2022). Guidelines for Safeguarding Good Research Practice. Code of Conduct. Available in German and in English.

[Goble 2014] Goble, C. (2014). Better software, Better Research. *IEEE Internet Computing*, 18(5):4–8.

[Goble et al. 2016] Goble, C., Howison, J., Kirchner, C., Nierstrasz, O., and Vinju, J. J. (2016). Engineering Academic Software (Dagstuhl Perspectives Workshop 16252). *Dagstuhl Reports*, 6(6):62–87.

[Gruenpeter et al. 2021] Gruenpeter, M. et al. (2021). Defining Research Software: a Controversial Discussion.

[Hannay et al. 2009] Hannay, J., MacLeod, C., Singer, J., Langtangen, H., Pfahl, D., and Wilson, G. (2009). How do scientists develop and use scientific software? In *ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 1–8.

[Hermann and Fehr 2022] Hermann, S. and Fehr, J. (2022). Documenting Research Software in Engineering Science. *Scientific Reports*, 12(6567).

[Hettrick et al. 2014] Hettrick, S. et al. (2014). UK Research Software Survey 2014, doi: 10.5281/zenodo.14809.

[Howison et al. 2015] Howison, J., Deelman, E., McLennan, M. J., Ferreira da Silva, R., and Herbsleb, J. D. (2015). Understanding the scientific software ecosystem and its impact: Current and future measures. *Research Evaluation*, 24(4):454–470.

[Howison and Herbsleb 2011] Howison, J. and Herbsleb, J. D. (2011). Scientific software production: incentives and collaboration. In *Proc. of the ACM 2011 conference on Computer supported cooperative work*, pages 513–522.

[Howison and Herbsleb 2013] Howison, J. and Herbsleb, J. D. (2013). *Incentives and integration in scientific software production*, pages 459–470. ACM.

[IEC 2014] IEC, I. (2014). ISO/IEC 25010: Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE. *Switzerland: ISO*.

[Jay et al. 2021] Jay, C., Haines, R., and Katz, D. S. (2021). Software Must be Recognised as an Important Output of Scholarly Research. *Int. Journal of Digital Curation*, 16(1):6. Number: 1.

[Jiménez et al. 2017] Jiménez, R. C., Kuzak, M., Alhamdoosh, M., Barker, M., Batut, B., Borg, M., Capella-Gutierrez, S., Chue Hong, N., et al. (2017). Four simple recommendations to encourage best practices in research software. *F1000Research*, 6:876.

[Katz 2014] Katz, D. (2014). Transitive credit as a means to address social and technological concerns stemming from citation and attribution of digital products. *Journal of Open Research Software*, 2(1).

[Katz et al. 2020] Katz, D., Chue Hong, N., et al. (2020). Recognizing the value of software: a software citation guide.

[Katz 2022] Katz, D. S. (2022). Towards sustainable research software.

[Katz et al. 2016] Katz, D. S. et al. (2016). Report on the third workshop on sustainable software for science: Practice and experiences (WSSSPE3). *CoRR*, abs/1602.02296.

[Kehrer and Penzenstadler 2018] Kehrer, T. and Penzenstadler, B. (2018). An exploration of sustainability thinking in research software engineering. In *Proc. of the 7th Int. Workshop on Requirements Engineering for Sustainable Systems (RE4SuSy 2018)*, volume 2223 of *CEUR Workshop Proc.*, pages 34–43. CEUR-WS.org.

[Knuth 1984] Knuth, D. E. (1984). Literate Programming. *The Computer Journal*, 27(2).

[Lamprecht et al. 2020] Lamprecht, A.-L. et al. (2020). Towards FAIR Principles for Research Software. *Data Science*, 3(1):37–59.

[Merali 2010] Merali, Z. (2010). Computational science: Error, why scientific programming does not compute. *Nature*, 467(7317):775–777.

[Momcheva and Tollerud 2015] Momcheva, I. and Tollerud, E. (2015). Software use in astronomy: an informal survey. *arXiv preprint arXiv:1507.03989*.

[Mourão et al. 2018] Mourão, B. C., Karita, L., and do Carmo Machado, I. (2018). Green and Sustainable Software Engineering – a Systematic Mapping Study. In *Proc. of the XVII Brazilian Symposium on Software Quality*, SBQS '18, page 121–130, New York, NY, USA. ACM.

[Munk et al. 2019] Munk, M., Koziar, K., Leinweber, K., Silva, R., Michonneau, F., et al. (2019). swcarpentry/git-novice: Software Carpentry: Version Control with Git, June 2019.

[Nenadic et al. 2022] Nenadic, A. et al. (2022). carpentries-incubator/python-intermediate- development: beta.

[Nieuwpoort and Katz 2023] Nieuwpoort, R. v. and Katz, D. S. (2023). Defining the roles of research software. *Upstream*.

[Rajlich and Bennett 2000] Rajlich, V. and Bennett, K. (2000). A staged model for the software life cycle. *Computer*, 33(7):66–71.

[Segal and Morris 2008] Segal, J. and Morris, C. (2008). Developing scientific software. *IEEE Software*, 25:18–20.

[Smith 2021] Smith, A. (2021). Enhanced support for citations on github.

[Smith et al. 2017] Smith, A. M. et al. (2017). Journal of open source software (JOSS): design and first-year review. *CoRR*, abs/1707.02264.

[Sufi et al. 2020] Sufi, S. et al. (2020). Report on the Workshop on Sustainable Software Sustainability 2019 (WOSSS19). Technical report, Zenodo.

[Terceiro et al. 2010] Terceiro, A., Costa, J., Miranda, J., Meirelles, P., Rios, L. R., Almeida, L., Flach, C., and Kon, F. (2010). Analizo: an extensible multi-language source code analysis and visualization toolkit. In *Brazilian Conference on Software: Theory and Practice (CBSoft) – Tools*, Salvador-Brazil.

[Toutain 2019] Toutain, T. G. L. d. O. (2019). Avaliação da estabilidade cerebral e conexões intra e inter-hemisféricas na modulação afetiva da dor. Master's thesis, Universidade Federal da Bahia, (PPGPIOS), Brasil.

[UNESCO 2021] UNESCO (2021). UNESCO Recommendation on Open Science.

[Venters et al. 2017] Venters, C. et al. (2017). Software sustainability: Research and practice from a software architecture viewpoint. *Journal of Systems and Software*, 138.

[Venters et al. 2021] Venters, C. et al. (2021). Software Sustainability: Beyond the Tower of Babel. In *BoKSS 2021*. Publisher: figshare.

[Venters et al. 2014] Venters, C., Jay, C., et al. (2014). Software sustainability: The Modern Tower of Babel. *CEUR Workshop Proceedings*, 1216:7–12.

[Versen 2020] Versen (2020). Manifesto on Software Research and Education in the Netherlands. versen.nl. `https://www.versen.nl/assets/manifesto/manifesto_software_onderzoekers_06-2.pdf`. [Accessed 10-Jun-23].

[Wilkinson et al. 2016] Wilkinson, M. D. et al. (2016). The FAIR Guiding Principles for Scientific Data Management and Stewardship. *Scientific Data*, 3.

[Wilson et al. 2017] Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., and Teal, T. K. (2017). Good enough practices in scientific computing. *PLOS Computational Biology*, 13(6):1–20.