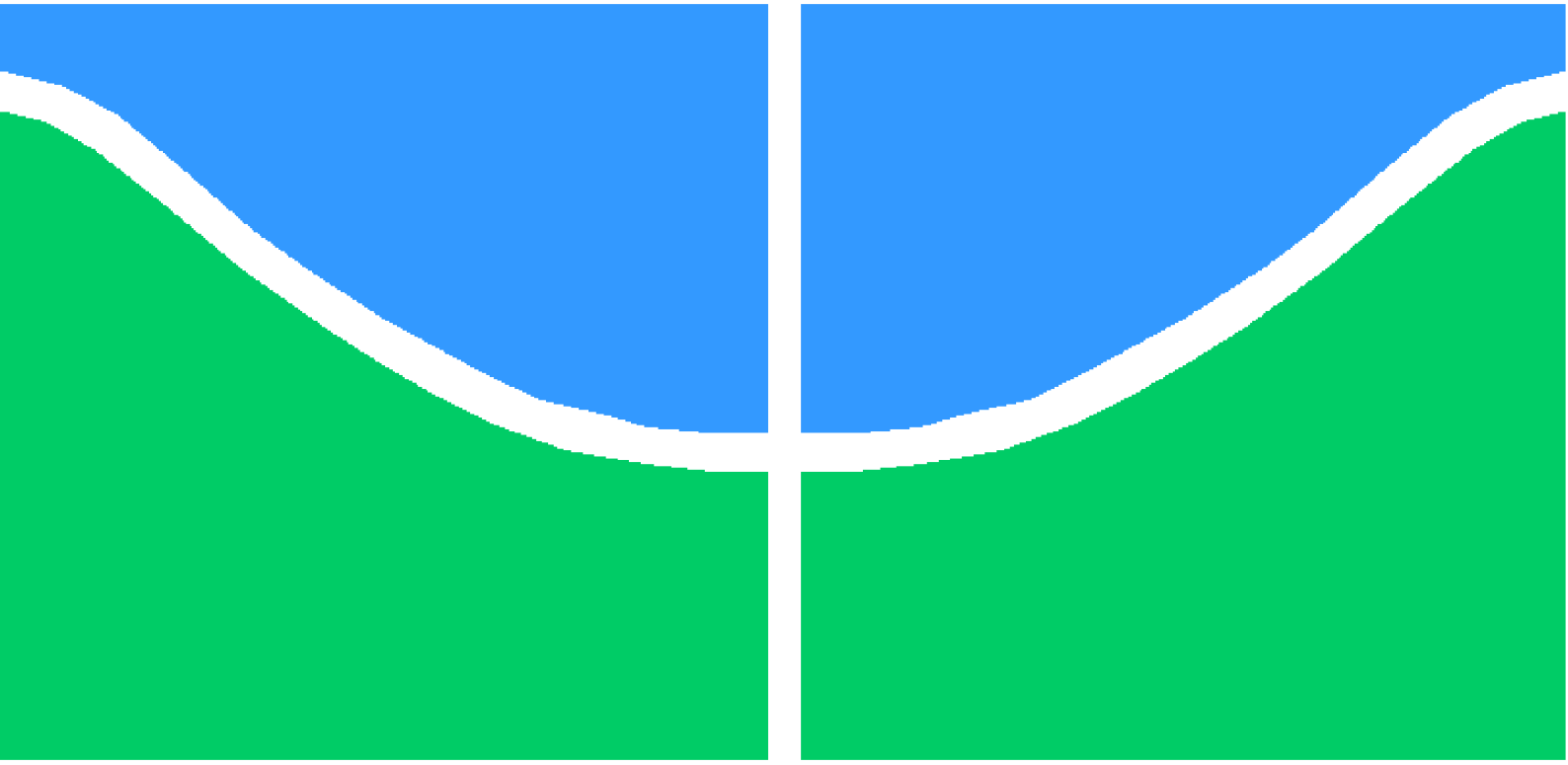


Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Fider Community: Ferramenta para Gestão de Solicitações de Funcionalidades

Autor: Bernardo Henrique Rosa Lima
Orientador: Prof. Dr. Renato Coral Sampaio

Brasília, DF
2021

Bernardo Henrique Rosa Lima

Fider Community: Ferramenta para Gestão de Solicitações de Funcionalidades

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software .

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Renato Coral Sampaio

Brasília, DF

2021

Bernardo Henrique Rosa Lima
Fider Community: Ferramenta para Gestão de Solicitações de Funcionalidades / Bernardo Henrique Rosa Lima . – Brasília, DF, 2021-
54 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Renato Coral Sampaio

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2021.

1. . 2. . I. Prof. Dr. Renato Coral Sampaio . II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Fider Community: Ferramenta para Gestão de Solicitações de Funcionalidades

CDU

Bernardo Henrique Rosa Lima

Fider Community: Ferramenta para Gestão de Solicitações de Funcionalidades

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software .

Trabalho aprovado. Brasília, DF, :

Prof. Dr. Renato Coral Sampaio
Orientador

Prof. Dr. André Barros de Sales
Convidado 1

Prof. Dr. Maurício Serrano
Convidado 2

Brasília, DF
2021

Agradecimentos

Não sei muito bem como começar essa parte, mas com certeza as primeiras pessoas que devo agradecer são os meus pais por sempre me apoiarem em todas as decisões da minha vida. Eles me deram o suporte emocional para passar por essa fase de Trabalho de Conclusão de Curso (TCC) e final de faculdade que com certeza foi uma das mais difíceis até agora, ainda mais durante esta época de pandemia.

Gostaria, também, reconhecer a ajuda da Clarissa Borges e Arthur Diniz por terem facilitado o contato com as comunidades. Além disso devo deixar meus agradecimentos aos às pessoas que se disponibilizaram para realizar as entrevistas, base do meu projeto: Sergio Durigan (Debian), Georges Gasile Stravacas (Gnome), Theo Reneck (Rocket.Chat), Cynthia Sanches (EOS Design System), Haydee Svab, Diego Oliveira e Leonardo Leite (Radar Parlamentar) e Tomaz Canabrava (KDE).

Finalizando esta breve sessão de agradecimentos, gostaria de deixar um recado para alguém que também esteja no momento final da faculdade e tendo dificuldades com o TCC: não se cobre tanto e tente levar essa fase com mais leveza, caso as coisas não deem certo e seja preciso deixar para o próximo semestre, está tudo bem (eu mesmo fiquei um ano em hiato na faculdade com apenas esse trabalho pendente). Vá no seu ritmo, faça terapia e tente não deixar a ansiedade causada pela pressão e confusão de publicar um primeiro trabalho tomar conta desse e outros aspectos da sua vida.

Espero que você sinta um alívio tão grande quanto o meu ao acabar essa fase.

"Ao ver o meu ego vir a falecer
Descubro o sentido da vida e ele é pra frente."
(Potyguara Bardo)

Resumo

A cooperação é um dos pilares do software livre. Há várias maneiras, não necessariamente técnicas, de se contribuir com projetos, sendo uma delas sugerir novas funcionalidades.

Escutar e entender os anseios de seus usuários é um fator chave para o sucesso de empresas, projetos e comunidades. Ainda sim, há falhas neste processo de comunicação.

Esta pequisa inicia o desenvolvimento de uma ferramenta para auxiliar comunidades de software livre no processo de gestão das solicitações de funcionalidades feitas pelos usuários.

Para isso foram realizadas as seguintes etapas: buscou-se as melhores práticas para auxiliar neste processo, tanto na literatura como analisando produtos disponíveis no mercado. Entrevistou-se comunidades, com intuito de levantar suas necessidades e pontos de melhoria. Em seguida foi iniciado o desenvolvimento de uma nova ferramenta, utilizando como base um dos produtos encontrados anteriormente.

Palavras-chave: software livre, comunidades de software livre, requisição de funcionalidades, ferramentas de requisição de funcionalidades

Abstract

This research seeks the best practices to assist in this process, both in the literature and analyzing different tools available on the market.

Then, through interviews with the communities, the reasons that lead to communication failures were explored, and a solution was proposed based on the information gathered.

Cooperation is one of free software's pillar. There are several ways, not necessarily technical, to contribute to projects, one of them being to suggest new features.

Listening and understanding the needs of your users is a key factor for the success of companies, projects and communities. Nevertheless, there are flaws in this communication process.

This research seeks to develop a tool to help open source communities on the users' feature request process.

For this, the following steps were carried out: the best practices were sought, both in the literature and analyzing products available on the market. Communities were interviewed in order to raise their needs and areas for improvement. Then, the development of a new tool was started, using one of the products previously found.

Key-words: open source, open source communities, feature requests, feature request tools

Lista de tabelas

Lista de abreviaturas e siglas

API - *Application Programming Interface*

BTS - *Bug Tracking System*

CSM - *Custom Service Manager*

FC - Fider Community

FOSS - *Free and Open-Source Software*

NPS - *Net Promoter Score*

Sumário

1 Introdução

O termo *Open Source* surgiu para classificar uma abordagem de desenvolvimento de software. Hoje refere-se à software, produtos ou projetos de código aberto e gratuito como FOSS, do inglês, *Free and Open-Source Software*, aqueles que seguem um conjunto de valores: troca de experiência, colaboração, transparência, desenvolvimento orientado para a comunidade, entre outros. [12]

Há várias maneiras de contribuir para um FOSS, entre elas: programar, documentar, sugerir novas funcionalidades, traduzir textos, participar de pesquisas, avaliá-lo, etc.

Duas plataformas que auxiliam neste processo são o GitHub e o GitLab, conhecidos por serem os principais serviços de hospedagem e versionamento de código. Elas permitem que usuários configurem repositórios públicos onde outros possam contribuir para o projeto não só tecnicamente, mas de diversas formas. Todas as contribuições são publicadas de forma visível, transparente e imputável.

Dado ao design, concepção e ferramentas destas plataformas, o GitHub e o GitLab são adequadas para FOSS, e por isso amplamente utilizados. Vários lideres de projetos indicam que a mudança de outros sistemas de versionamento para estas foi o ponto principal para aumento de contribuidores, dadas às suas facilidades [7].

Para melhorar este processo de contribuição, a própria comunidade criou padrões para que seus membros pudessem apoiar-se de forma organizada e transparente. Com isso foram adotados vários conceitos e ferramentas como *gitflow*, *issues*, zenhub e etc.

Mesmo sendo ferramentas e conceitos amplamente utilizados e difundios no meio da ciências da computação [4], não é correto esperar que usuários tenham que aprender sobre estes para que possam contribuir aos FOSS. Afinal, mesmo se tratando das ferramentas amplamente usadas, como Github e Gitlab, os usuários não-técnicos e iniciantes ainda apresentam dificuldades ao utilizá-las. [7].

Atualmente, para que alguém possa sugerir novas funcionalidades, é necessário ter conhecimento sobre issues, labels, markdown, templates e da interface das plataformas. Para certos usuários pode ser um empecilho na hora de começar a contribuir, acarretando em desistência.

Sendo a colaboração um dos principais fatores para o sucesso dos FOSS [10], é necessário pensar em uma nova maneira para que estes usuários possam sugerir e acompanhar novas funcionalidades, dar *feedbacks* e interagir com a comunidade.

Dentro do contexto desta proposta, tem-se a seguinte questão de pesquisa:

"Como facilitar a sugestão e o acompanhamento de novas funcionalidades em projetos de software livre?"

1.1 Objetivos

1.1.1 Objetivo Geral

Este trabalho tem como objetivo geral propor um processo para sugestão de funcionalidades em projetos de comunidades de software livre.

1.1.2 Objetivos Específicos

Foram definidos os seguintes objetivos específicos para que o objetivo geral seja alcançado:

- Identificar padrões adotados no processo de sugestão de funcionalidades.
- Analisar as principais ferramentas no mercado focadas em neste processo e mapear suas *features*.
- Investigar os principais problemas de comunicação entre as comunidades de software aberto e seus usuários.
- Identificar as suas necessidades neste processo.
- Implementar plataforma para solução de alguns dos problemas encontrados.

1.2 Organização do Trabalho

Este trabalho de conclusão de curso está organizado nos seguintes capítulos:

- Capítulo 1 - Introdução: Apresentação breve da problemática e esclarecimento dos objetivos do trabalho.
- Capítulo 2 - Referência Teórico: Contextualização dos assuntos abordados.
- Capítulo 3 - Metodologia: Apresentação da metodologia de pesquisa e descrição da sua aplicação.
- Capítulo 4 - Pesquisa: Análise da literatura e das ferramentas.
- Capítulo 5 - Entrevistas: Apresentação do processo e resultados das entrevistas.

- Capítulo 6 - Desenvolvimento: Proposta de solução e especificação dos requisitos do Fider Community e apresentação do desenvolvimento do projeto.
- Capítulo 7 - Considerações Finais.

2 Referencial Teórico

2.1 Software de Código Aberto Gratuitos

Software fechado é a denominação usada para softwares que apenas uma pessoa, time ou organização é proprietária e mantém controle exclusivo em relação à ele. Sendo assim, usuários externos não possuem acesso ao código fonte, e para utilizá-los devem concordar com as condições imposta pelo(s) dono [12].

Como já dito, FOSS é um termo que classifica uma abordagem de desenvolvimento orientada à comunidade, onde os seus contribuidores possuem acesso ao código fonte e podem utilizá-lo ou modificá-lo de acordo com sua licença.

As licenças são termos de uso de softwares *open source*, também conhecidos como *copyleft*, e determinam a forma que estes podem ser usados, modificados e distribuídos. Elas são desenhadas de forma a encorajar os desenvolvedores a contribuir para o projeto [2].

Há licenças mais liberais, como a MIT, que basicamente permite o uso e alteração do software para quase qualquer propósito. E há outras mais restritivas, como a GNU GPLv3, que não permite que softwares que usufruem do seu código fonte sejam proprietários [1].

2.2 Sugestões de Funcionalidades

O envolvimento do usuário como parte do desenvolvimento de produtos de software é um tema tratado pela literatura há muito tempo. Pelo fato de terem adquirido o conhecimento sobre o produto apenas pelo seu uso, estes possuem uma perspectiva diferente dos pesquisadores e programadores [6].

Tal perspectiva é fundamental, afinal novas funcionalidades que não refletem os interesses de seus utilizadores costumam acarretar em tempo perdido no desenvolvimento, aumento de complexidade e custos adicionais. E mesmo que a engenharia de software, mais especificamente a de requisitos, defina metodologias e diretrizes de como informações devem ser coletadas, as informações oferecidas não costumam ser padronizadas [9].

O estado atual da indústria de software é caracterizado por ciclos rápidos de desenvolvimento, e neste contexto a entrada de informações dos usuários precisou se tornar cada vez mais intensa e dinâmica [14]. Sendo assim, cada vez mais empresas estão ten-

tando engajar seus usuários a participar ativamente de coleta de *feedbacks*, sugestões de melhorias e levantamento de novas ideias.

2.3 Sugestões de Funcionalidades em Softwares de Código Abertos Gratuitos

De acordo com Gassmann e Enkel [3], FOSS são grandes exemplos de revolução da maneira que o processo de inovação sempre foi pensado. É um processo de desenvolvimento de software por programadores independentes que, de forma cooperativa, se juntam para desenvolver produtos que competem com de grande empresas.

Porém, enquanto a coordenação e distribuição de trabalhos entre desenvolvedores nestes projetos é muito bem estabelecida; utiliza-se ferramentas de versionamento de código, kanbans, repositórios, lista de email, metodologias, entre outros. Poucas práticas são adotadas para apoiar a comunicação entre membros da comunidade (majoritariamente técnicos) e usuários não-técnicos durante o processo de requisitar melhorias aos desenvolvedores [13].

Mesmo quando há um meio de comunicação, não é garantia de sucesso. De acordo com Heck e Zaidman [5], mesmo quando FOSS possuem uma plataforma para esta comunicação, ainda há problemas como: repetições excessivas de requerimento de novas funcionalidades, requisição de funcionalidades já existentes e assegurar que as solicitações dos usuários são válidas.

Há a necessidade de entender e estruturar como deve funcionar este processo, de forma que o relacionamento entre os membros da comunidade se torne mais produtivo.

3 Metodologia

Inicialmente foi pesquisado como é feita a sugestão de funcionalidade em projetos, produtos e empresas (não apenas FOSS). Esta etapa foi dividida nas seguintes atividades:

1. Procurar na literatura se há padrões, normas ou regras de como deve ser feita a coleta de sugestão de funcionalidades por usuários. Para esta etapa foram utilizadas as bases de dados SCOPUS (Elsevier), ACS Journals Search e Scielo.

2. Selecionar e analisar diferentes ferramentas de sugestão de funcionalidades do mercado, mapeando suas principais características.

Em um segundo momento, foi produzido um roteiro semi-aberto de entrevistas para descobrir quais são os maiores problemas dos FOSS em relação à comunicação com seus usuários, focando principalmente no processo de sugestões de funcionalidades. Este questionário foi aplicado à mantenedores e contribuidores de projetos de código aberto.

A terceira etapa do projeto foi utilizar os dados levantados tanto pela pesquisa quanto pelas entrevistas para propor e desenvolver uma solução adequada às necessidades das comunidades. Também foram pesquisados padrões de design de telas para propor uma solução orientada aos usuários.

4 Pesquisa

4.1 Análise da Literatura

A evolução de softwares é uma atividade inevitável. Em projetos de código aberto costuma-se utilizar uma maneira menos formal para a coleta de requisitos, onde são representados como histórias de usuários e possuem foco na evolução do projeto. Estes modelo é reconhecido como Engenharia de Requisitos *Just-In-Time*.

Em FOSS com grande quantidade de usuários, muitos se envolvem e solicitam novas funcionalidades, porém não há nenhum controle em relação ao que é inserido, acarretando em problemas de repetição de solicitações. Os responsáveis por analisar, filtrar, priorizar e decidir se estas solicitações serão implantadas ou não, são, geralmente, os próprios desenvolvedores, que despendem de muito tempo com esta atividade.

Heck e Zidman [5] procuraram identificar os motivos que levam à esta repetição de solicitações. Eles analisaram repositórios de vinte FOSS e levantaram que mesmo projetos maduros, como Mozilla Firefoz, chegam a ter mais de 35% de suas *issues* marcadas como duplicada. Com isso, identificaram que as prováveis causas são: usuários não conseguem identificar facilmente que a funcionalidade já existe no sistema ou ela já foi solicitada. Para diminuir as repetições, fizeram algumas recomendações:

- Deve-se sugerir ao usuário que pesquise e procure tirar dúvidas (por lista de e-mail/discussão, ou outro canal de contato) antes de inserir novas requisições;
- Devem haver diretrizes claras de quais informações devem ser apresentadas na solicitação;
- Não deve ser aceito código-fonte;

Outro problema comum é o fato de softwares não possuírem canais ou plataformas que auxiliem na distinção entre requisições de feature, reporte de bugs e *feedbacks* gerais sobre o produto [11] [8] .

Maalej e Nabil [8] apresentam um estudo sobre este problema no âmbito de aplicações móveis, que geralmente possuem como único canal de comunicação a sessão de comentários do aplicativo na loja. Foi identificado que há um grande volume de *feedbacks* inseridos diariamente (podendo chegar a centenas no caso de aplicações mais populares), com expressiva repetição e baixa qualidade de informações inseridas.

Merten et al [11] também identificaram que os mesmos problemas também podem ocorrer em plataformas especializadas em coleta de issues (como o Bugzilla). Em ambos os casos foram propostos modelos de classificação de *feedbacks* utilizando técnicas como análise de metadados, frequência de palavras, regras de linguística e análise de sentimento.

4.2 Análise de Ferramentas

Sendo um problema conhecido, é de se esperar que haja ferramentas no mercado que ajudem nestas necessidades. Foram analisadas nove destas, e em sua maioria foram encontradas funcionalidades básicas para atingir o objetivo: usuários podem adicionar, comentar e votar em *features*, além de visualizar solicitações (tanto suas quanto de outros usuários).

Abaixo será apresentada uma breve descrição sobre as ferramentas, suas funcionalidades e uma planilha comparativa destas.

4.2.1 Instabug - Feature Request Management

Instabug é uma ferramenta focada em projetos mobile, que possui quatro frentes de serviço: reporte de falhas e quebras da aplicação, monitoramento de desempenho e pesquisa para coleta de *feedback* dos usuários.

A pesquisa é integrada na aplicação, e pode ser customizada de acordo com diferentes perfis de usuário. Além disso abrange várias vertentes: avaliação das funcionalidades específicas já implementadas, NPS (Net Promoter Score), percepção da marca, entre outros.

Além de apresentar as solicitações feitas pelos usuários, o Instbug também possui integração com ferramentas de controle do fluxo de trabalho da equipe de desenvolvimento (como Github, Gitlab, Jira, Trello, etc). Assim, o usuário pode manter acompanhar o status das funcionalidades: se estão em análise, programadas, sendo desenvolvida ou finalizadas.

A ferramenta também permite contato direto com o usuários para que os a empresa possa solicitar mais informações sobre alguma solicitação que foi recebida ou *feedback* fornecido.

4.2.2 Canny - Feature Request Tracking

Canny é uma ferramenta especializada no rastreamento de solicitações de funcionalidades. Além de coletá-las dos usuários, ela também propõe coletar solicitações do próprio time.

Além das funcionalidades básicas, ela também possui integração com ferramentas de acompanhamento do *status* de andamento das solicitações e permite contato direto com usuários.

Como diferencial, apresenta detalhadamente o *roadmap* de implementação das novas funcionalidades para que os usuários possam saber com maior precisão quando estarão disponíveis.

4.2.3 Automation Hub (AU)

Automation Hub é uma ferramenta da UIPath, empresa especializada em *Robotic Process Automation* (RPA). A plataforma permite que os usuários das automações sugiram novos processos a serem automatizados, sendo específica para esta tecnologia.

Esta ferramenta possui todas as funcionalidades do Instabug, além várias outras não tão significativas para esta pesquisa, como: mapa de calor de automações, painel de custos, geração automática de documentação, entre outros.

4.2.4 Feathub

Feathub é uma ferramenta *Open Source* e gratuita. É integrada com o Github e possui as funcionalidades básicas citadas acima. É a ferramenta mais simples das pesquisadas.

4.2.5 Fider

Fider também é um FOSS, como a anterior. É uma plataforma web que das funcionalidades básicas, apenas não permite que usuários vejam o histórico de solicitações que fizeram.

Como diferencial, a ferramenta permite que os usuários se inscrevam em tópicos para que sejam informados de atualizações sobre a requisição de funcionalidade (caso seja comentada ou tenha mudança de status).

Também possui um sistema de *tags* no qual os administradores do sistema podem classificar as solicitações para melhor organização interna.

Além disso, também possui um sistemas de APIs públicas, para que as informações coletadas pelo sistema possam ser utilizadas como o time preferir.

4.2.6 Feature Upvote (FU)

Feature Upvote também é uma plataforma web que possui as mesmas funcionalidades Feathub, e como diferencial, ele possibilita que sejam criados quadros privados para que grupos específicos de usuários possam separar suas ideias dos demais.

4.2.7 Demais Softwares

Foram encontrados alguns outros softwares que atendem ao objetivo: UserVoice , HelloNext e Savio. Todos são privados.

Estes não serão apresentados à frente na tabela comparativa pois apresentam as mesmas funcionalidades encontradas nos softwares citados anteriormente.

4.3 Funcionalidades Encontradas

Abaixo será apresentada uma tabela com os nomes e descrições das funcionalidades que foram encontradas nas ferramentas citadas. Depois, será apresentado uma tabela comparativa de funcionalidades entre elas.

É importante ressaltar que foram apresentadas apenas as funcionalidades relevantes para o objetivo proposto.

4.3.1 Descrição das Funcionalidades

Nome	**Descrição**
Adicionar Solicitação	Usuário pode adicionar nova solicitação que deseja fazer ao projeto.
Comentar Solicitação	Usuários podem comentar solicitações, tanto suas quanto de outros usuários.
Visualizar suas Solicitações	Usuário pode visualizar suas próprias solicitações.
Votar em Solicitações	Usuários podem votar em solicitações que desejam ter mais destaque
Sistema de Tagueamento	Usuários podem utilizar etiquetas para classificar suas solicitações.
Apresentar Status da Solicitação	Apresenta se a feature foi aceita e seu andamento (Ex.: Aceita, Em Análise, Recusada, Finalizada, etc).
Integração com Ferramentas de Workflow	Integração com ferramentas que mapeiam as solicitações com issues em ferramentas de trabalho do time de desenvolvimento (Ex.: Github, Gitlab, Jira, Trello, etc).
Apresentar Solicitações Similares	Quando uma solicitação é inserida, são apresentadas outras relacionadas ao pedido. Essa função tem o intuito de diminuir a duplicação de requisições.
Contato Direto com o Usuário	Os administradores do sistema podem se comunicar diretamente com os usuários pela aplicação.
Roadmap de Implementação	O sistema apresenta o roadmap de implementação das features para os usuários, utilizando como base a ferramenta de workflow escolhida.
Inscrição em Solicitação	Permite que os usuários se inscrevam em solicitações para que sejam notificados quando houver atualizações nesta.
APIs	Sistema de APIs que permitem que as informações da aplicação sejam acessadas por endpoints específicos.
Quadros Privados	Permite a criação de quadros privados que são acessados apenas por grupos de usuários autorizados.

Tabela 1 – Tabela de Descrição das Funcionalidades

4.3.2 Tabela Comparativa de Funcionalidades

Funcionalidade	**Instabug**	**Canny**	**AU**	**Feathub**	**Fider**	**FU**
Adicionar Solicitação						
Comentar Solicitação						
Visualizar suas Solicitações						
Votar em Solicitações						
Apresentar Status da Solicitação						
Sistema de Tagueamento						
Integração com Ferramentas de Workflow						
Apresentar Solicitações Similares						
Contato Direto com o Usuário						
Roadmap de Implementação						
Inscrição em Solicitação						
APIs						
Quadros Privados						

Tabela 2 – Tabela Comparativa de Funcionalidades

5 Entrevistas

Com intuito de explorar melhor os problemas das comunidades, foram feitas entrevistas com participantes destas. As entrevistas procuraram descobrir problemas não só relacionados diretamente à solicitações de funcionalidades, mas também em todo o âmbito da comunicação com os usuários finais.

As entrevistas podem ser encontradas, na íntegra, no apêndice Entrevistas.

5.1 Processo

As entrevistas foram realizadas entre os meses de março e abril de 2021 com representantes de seis comunidades, sendo elas:

- Debian - Sistema operacional.
- Gnome - Ambiente de trabalho e plataforma de desenvolvimento.
- Rocket.Chat - Plataforma de Comunicação.
- EOS Design System - Plataforma de Design System.
- Radar Parlamentar - Aplicativo que ilustra as semelhanças entre partidos políticos com base na análise matemática dos dados de votações que ocorrem na casa legislativa.
- KDE - Comunidade internacional de software livre que produz um conjunto de aplicativos multiplataforma.

Para que as informações fossem mais precisas e refletissem melhor a realidade dos projetos, foram realizadas entrevistas com participantes que possuíssem sólida experiência de como a comunicação com a comunidade funciona.

É importante ressaltar que mesmo as entrevistas sendo feitas com contribuidores experientes das comunidades, suas visões não a representam como um todo.

5.2 Resultados

Antes de apresentar os resultados, é importante pontuar que das comunidades citadas, a Rocket.Chat se destoa em alguns pontos: é uma empresa, e não uma fundação. Enquanto o seu projeto base se enquadra como FOSS, há uma versão privada do projeto

que é disponibilizada apenas por clientes pagantes da empresa, que não se enquadra e não será considerada para o estudo.

5.2.1 Comunidades e suas Diferentes Realidades

É importante esclarecer que e o tamanho das comunidades variam, e muito. Um dos fatores é a quantidade de projetos: Rocket.Chat, EOS e Radar possuem um projeto principal, enquanto KDE, Debian e Gnome possuem vários. Essas últimas muitas vezes são descritas como um guarda-chuvas de projetos.

E mesmo entre comunidades com apenas um projeto principal, o tamanho da comunidade que contribui ativamente pode variar muito. Sendo assim, a realidade, os desafios e os seus problemas são diferentes.

5.2.2 Utilização de Softwares Livres

Todas as comunidades, menos a Rocket.Chat, apresentaram a preocupação em utilizar softwares livres. Seja para comunicação, acompanhamento do desenvolvimento, rastreamento de bugs ou quaisquer outras necessidades. O Debian, por exemplo, relatou que mesmo que vários de seus usuários utilizem Telegram (um FOSS), a comunidade não pretende adotá-lo como canal de comunicação oficial por seus servidores serem fechados.

Sendo assim, é de crucial importância que qualquer solução proposta para solucionar algum problema das comunidades seja também um FOSS, e de preferência que uma instância possa ser configurada pela própria.

5.2.3 Canais de comunicação

A comunicação é feita de forma diferente em cada comunidade, sendo os únicos pontos em comum é que todas utilizam Github ou Gitlab, e quase todas utilizam listas de discussões, com exceção da Rocket.Chat.

Comunidades maiores e mais antigas, como KDE, Debian e Gnome, possuem muitos canais de comunicação, mas o processo de reconhecê-los como oficiais é diferente. Algumas relatam problemas com o modelo de comunicação usado:

O Debian define que os canais de comunicação oficiais são apenas IRC e lista de discussões por e-mail. Eles reconhecem que estas plataformas não são atraentes para usuários novos, sendo usada principalmente pelos contribuidores mais antigos.

Hoje a maior parte dos usuários está no Telegram, exatamente por ser uma ferramenta que proporciona uma experiência melhor. Dado este fato, há um movimento para a mudança de suas plataformas atuais, mesmo que lento.

Já a Gnome utiliza várias plataformas diferentes: Matrix, Telegram, Discourse, Rocket.Chat, IRC e listas de discussões. Eles enxergam essa comunicação fragmentada em vários canais como um problema, que resulta na perda de informação.

Há várias pessoas trabalhando para tentar resolver este problema da comunicação. Recentemente fizeram uma pesquisa com os contribuidores recorrentes da comunidade para saber quais features elas precisam dos canais para que sejam implementadas no Matrix, e migrar de uma vez.

A KDE, por sua vez, considera qualquer canal que seja aberto por um desenvolvedor como oficial. E possuem vários em diferentes plafataformas: Redit, Facebook, Telegram, mIRC, Matrix e Discord. E mesmo com tantos meios, não foi relatado nenhum problema com a comunicação.

5.2.4 Planejamento Estruturado de Entregas

Não é comum que as comunidades possuam um planejamento estruturado para entregas, roadmaps e nem gerentes de produto propriamente ditos. As únicas que possuem são Rocket.Chat e EOS Design System.

A EOS reconhece que é uma comunidade não convencional nesse sentido. Eles possuem uma pessoa que ocupa o cargo de gerente de produtos que é responsável por coletar todos os feedbacks da comunidade, priorizá-los e decidir o que entrará, ou não, no roadmap do próximo semestre, para que seu time principal implemente.

A tomada de decisões é centralizada , onde eles possuem bem definido o que será feito, quais problemas querem resolver e os grupos que querem focar. Eles acreditam que esse é um dos elementos de sucesso do projeto.

Essa centralidade de tomada de decisões vem do fato que o projeto se originou de um produto privado, que depois se tornou aberto. É importante ressaltar que eles possuem uma comunidade, que mesmo pequena, é ativa e auxilia tanto na execução do roadmap como também implementam funcionalidades que os interessam.

A Rocket.Chat, por ser uma empresa, possui uma pessoa que ocupa o cargo de gerente de produtos que também é responsável por organizar todas as solicitações, vindas da comunidade mas principalmente de clientes pagantes, e construir o roadmap para ser seguido pelo time interno.

Um problema relatado, dada essa dinâmica incomum, foi que ainda não há transparência em relação ao que será feito pelo time interno e o que está aberto para comunidade implementar. Isto gera dois problemas: os trabalhos podem se sobrepor e a comunidade pode implementar funcionalidades que estão disponíveis apenas na versão paga, que não serão aceitas.

5.2.5 Estrutura Descentralizada de Contribuições

Comunidades maiores, como Debian, Gnome e KDE, apresentam um sistema descentralizado de contribuições. Isso significa que o(s) mantenedor(es) dos projetos são responsáveis por determinar o que será implementado ou aceito.

Os entrevistados salientaram que é comum que pessoas externas às comunidades vejam cada uma como um bloco unificado de pessoas que seguem os mesmos protocolos. Mas olhando por dentro é muito mais individualizado. Há um conjunto básico de valores que é compartilhado, mas não existe conjuntos de regras como cada projeto ou mantenedor deve seguir.

5.2.6 Solicitações de melhorias - Geral

O processo de requisição de *features* é bem diferente entre as comunidades. Mas todas possuem um ponto em comum: nenhuma utiliza ferramentas unicamente para este intuito.

As comunidades menores, por recebem menos solicitações, não relataram problemas em receber e organizar suas solicitações. O Radar utiliza o quadro de *issues* do Gitlab, que é utilizado principalmente pelos mantenedores do projeto.

A EOS utiliza o Trello, ferramenta que auxilia na criação de um Kanban, que a comunidade e seus usuários tem acesso. Novas ideias costumam ser discutidas pelo Slack, maturadas e depois cadastradas no *backlog*.

A Rocket.Chat possui um time de pessoas que são especializadas em receber estas solicitações, as quais podem ser feitas via diferentes canais: Gitlab, Fórum, CMS, reuniões periódicas com os clientes, entre outros. O objetivo deste time é organizar e clusterizar essas informações para que, independente do canal, os dados sejam padronizados e passados ao gerente de produto.

Os projetos do Gnome não possuem um canal oficial ou forma definida para que usuários ou membros da comunidade relatem bugs e solicitem novas features. Sendo assim, os mantenedores dos projetos são repensáveis por definir como serão estes processos.

5.2.7 Solicitações de melhorias - Debian e KDE

A comunidade Debian utiliza um software chamado Bug Tracking System (BTS), tanto para bugs como solicitações e novas ideias. O BTS é um software que foi desenvolvido por um dos líderes do projeto Debian em 1994 e sofreu pouquíssimas alterações até hoje.

Ele é similar ao Bugzilla, sendo um pouco peculiar: funciona basicamente por e-mail. Mesmo tendo uma interface web,ela é apenas para leitura, não sendo possível mani-

pular qualquer informação (abrir bugs, fechá-los, adicionar tags, etc). Para a manipulação de qualquer informação é necessário utilizar e-mails.

A comunidade está acostumada com a ferramenta, mas reconhece que a dificuldade que é imposta aos usuários para realizar ações simples é muito alta. Para reportar um novo bug, por exemplo, o usuário tem que enviar um e-mail para o endereço do BTS com as tags específicas das informações necessárias para cadastrá-lo (id do pacote, versão, tags do bug, etc).

Não é um processo simples para um usuário que não conhece muito do software. Mesmo havendo *wikis* para ajudar nesse processo, reconhecem que caso um usuário possua algum problema, não vai querer perder horas aprendendo como reportá-lo corretamente.

A KDE, do mesmo modo, apresenta problemas nesta área. Eles utilizam o Bugzilla, ferramenta que para os padrões de hoje deixa a desejar na questão de usabilidade. Inclusive os membros da comunidade tem o costume de abrir os bugs/*features requests* para que os usuários não tenham que lidar com a ferramenta.

O maior problema apresentado quando é debatida a troca das ferramentas, por outras que possuem melhor usabilidade, é o histórico. Ambas possuem mais de 25 anos de discussões de problemas e propostas de melhorias, e perder estas informações não é uma opção.

5.3 Resultados

Foram levantados diferentes problemas entres as comunidades entrevistadas. Dado o fato da KDE e Debian serem as únicas que relataram problemas no processo de requisição de solicitação, a proposta será focada nelas e na sua necessidade de adotar uma ferramenta com interface mais amigável.

Considerando a preocupação destas em utilizar apenas softwares livres, as únicas ferramentas que se encaixam são FeatHub e Fider. Entre estas duas, a que apresenta mais funcionalidades necessárias para atender estas comunidades é a Fider, que mesmo sendo um projeto relativamente novo, apresenta uma comunidade ativa e lançamentos frequentes de novas versões.

Mesmo possuindo mais funcionalidades, a ferramenta ainda não possui a capacidade de atender todos os pontos levantados pelas comunidades, sendo assim, ainda há necessidades que devem ser atendidas antes que possa ser adotada.

5.4 Histórico

Ambas as comunidades utilizam as mesmas ferramentas (BTS e Bugzilla) há mais de 25 anos, e nelas está presente o histórico de todas as solicitações já realizadas. Este é riquíssimo em informações, contendo: dores dos usuários, suas requisições, discussões, ideias de solução, problemas em soluções adotadas, etc.

Além de ser usado para contextualizar problemas atuais, também representa parte da história destas comunidades. Sendo assim, tanto a comunidade do Debian quanto do KDE apontaram que um grande problema para utilizar outras plataformas é separar este histórico das novas informações que entrariam.

Posto isso, existe a necessidade de transferir este histórico para o Fider. É importante frisar que seriam necessárias soluções diferentes para cada comunidade, já que utilizam ferramentas diferentes.

5.5 Suporte à Múltiplos Projetos

Como dito anteriormente, estas comunidades agrupam dezenas de projetos diferentes. Atualmente, o Fider suporta apenas um, e não possui maneira eficiente de separar solicitações de projetos diferentes.

Uma alternativa seria criar uma instância diferente do Fider para cada projeto, porém além de resultar em um processo custoso para os usuários se cadastrarem em todos os projetos que procuram contribuir, os dados estariam descentralizados, dificultando o acesso e pesquisas.

5.6 Ferramenta de Pesquisa

Um dos pontos levantados é o fato que o Bugzilla e o BTS possuem ferramentas de pesquisa robustas, diferente da maior parte dos softwares que foram avaliados por estas comunidades.

O Fider, em específico, possui uma ferramenta de pesquisa simples, que não abrange todas as necessidades das comunidades.

6 Desenvolvimento

Com o intuito de solucionar apenas a segunda necessidade citada, suporte à múltiplos projetos, foi desenvolvida uma aplicação web responsável por agrupar diferentes instâncias do Fider: Fider Community (FC). Esta traz as seguintes soluções:

- **Agrupar Projetos:** A plataforma deve permitir que mantenedores vinculem as instâncias do Fider de seus projetos, de forma que sejam apresentados de forma centralizada e pesquisável para os usuários.
- **Unificação do Processo de Autenticação:** A plataforma deve assegurar que o usuário precise se autenticar apenas uma vez e tenha acesso à todos os projetos cadastrados em uma comunidade.
- **Acesso Facilitado aos Dados:** O Fider possui um conjunto de APIs para acessar os dados de um projeto. A plataforma deve permitir que os dados de todos, associados à uma comunidade específica, sejam acessados de forma centralizada.

As necessidades apresentadas acima foram descritas em requisitos, separados em épicos, funcionalidades e tarefas, sendo eles:

6.1 Agrupar Projetos (E1)

6.1.1 Gerenciar Projetos no Fider Community (E1F1)

- Adicionar Projetos (E1F1T1): deve permitir que os mantenedores da comunidade enviem um e-mail aos mantenedores dos projetos solicitando que estes sejam adicionados à comunidade;
- Remover Projetos (E1F1T2): deve permitir que os mantenedores removam projetos adicionados;
- Apresentar Projetos (E1F1T3): deve permitir que os mantenedores apresentem aos usuários todos os projetos;

6.1.2 Adicionar Projeto ao Fider Community (E1F2)

- Aceitar Convite (E1F2T1): deve permitir que os mantenedores dos projetos aceitem que seus projetos sejam adicionados à comunidade;

6.2 Unificação do Processo de Autenticação (E2)

6.2.1 Gerenciar Processo de Autenticação (E2F1)

- Extrair processo de autenticação (E2F1T1): atualmente este processo do Fider é realizado na própria aplicação, porém para que seja possível unificá-los será necessário um serviço separado;
- Unificar autenticação entre comunidades e instâncias (E2T1T2): deve ser possível cadastrar-se e autenticar-se apenas uma vez;

6.3 Acesso Facilitado aos Dados (E3)

6.3.1 Apresentar Funcionalidades (E3F1)

- Apresentar solicitações de funcionalidades das comunidades (E3F1T1): apresentá-las, assim como outras informações: tags, usuários solicitantes e resumo.
- Pesquisar solicitações (E3F1T2): por projeto, título e tags.

6.3.2 Unificar APIs (E3F2)

- Disponibilizar APIs (E3F2T1): disponibilizar acessos às informações, via API, que já estão presentes no Fider. Estas podem ser encontradas na documentação.

6.4 Cadastro de Mantenedores (E4)

6.4.1 Cadastrar Mantenedores (E4F1)

- Cadastrar Mantenedores (E4F1T1): estes poderão cadastrar e deletar novos projetos.
- Autenticar Mantenedores (E3F1T2): devem realizar login para terem acesso às funcionalidades.

6.5 Cronograma

O cronograma de desenvolvimento foi inicialmente planejado e distribuído para 14 semanas, dos dias 19/07/2021 à 31/10/2021. Porém, por questões de saúde, o inicio do projeto foi atrasado, sendo inciado apenas no dia 30/08/2021. Com isso, o cronograma real possui apenas 9 semanas.

Antes de definir os requisitos formalmente e da mudança de calendário, foram planejas algumas atividades:

1. Estudo da ferramenta Fider e discussão da proposta com os mantenedores.
2. Implementar, no Fider, a vinculação ao Fider Community.
3. Implementar interface de comunidades vinculadas no FC.
4. Agrupar e disponibilizar APIs das diferentes instâncias do Fider.
5. Implementar processo de autenticação no FC.
6. Compartilhar sessão com Fider vinculados.

Este planejamento sofreu muitas mudanças, não só pelo fato do inicio tardio mas também por outras questões: a primeira atividade sugeria que a solução do FC fosse discutido com mantenedores e desenvolvedores do Fider, para que ambas se complementassem. Porém o contato não foi bem sucedido, já que nenhuma resposta foi obtida por e-mail, issues ou no próprio fórum da comunidade.

Além disso, ambos os fatores citados também impossibilitaram a implementação de um sistema unificado de autenticação (E2), já que seria necessário mais tempo e um conhecimento aprofundado da arquitetura do Fider.

Após estudo da ferramenta, também foi constatado que a API do Fider é aberta, e todas as informações estão disponível para leitura sem necessitar de credenciais de acesso. Assim, tornou-se desnecessário que os mantenedores do Fider concedam acesso (E1F2).

Com essas mudanças, não foram necessárias alterações no projeto do Fider. Abaixo encontram-se as atividades realizadas e o cronograma real do projeto.

1. Estudo da ferramenta Fider.
2. Cadastro de mantenedores.
3. Autenticação de mantenedores.
4. Cadastro, remoção e apresentação de projetos.
5. Agrupar dados dos projetos para disponibilizar features.
6. Filtrar features por tags, titulo e projeto.
7. Reporte dos resultados.

Tabela 3 – Cronograma de atividades

Atividades	Semanas								
	01	02	03	04	05	06	07	08	09
1									
2									
3									
4									
5									
6									
7									

6.6 Tecnologias Utilizadas

O *frontend* seguiu o padrão atual do Fider, sendo desenvolvido utilizado ReactJS, um framework Javascript para construção de aplicações web, com Typescript.

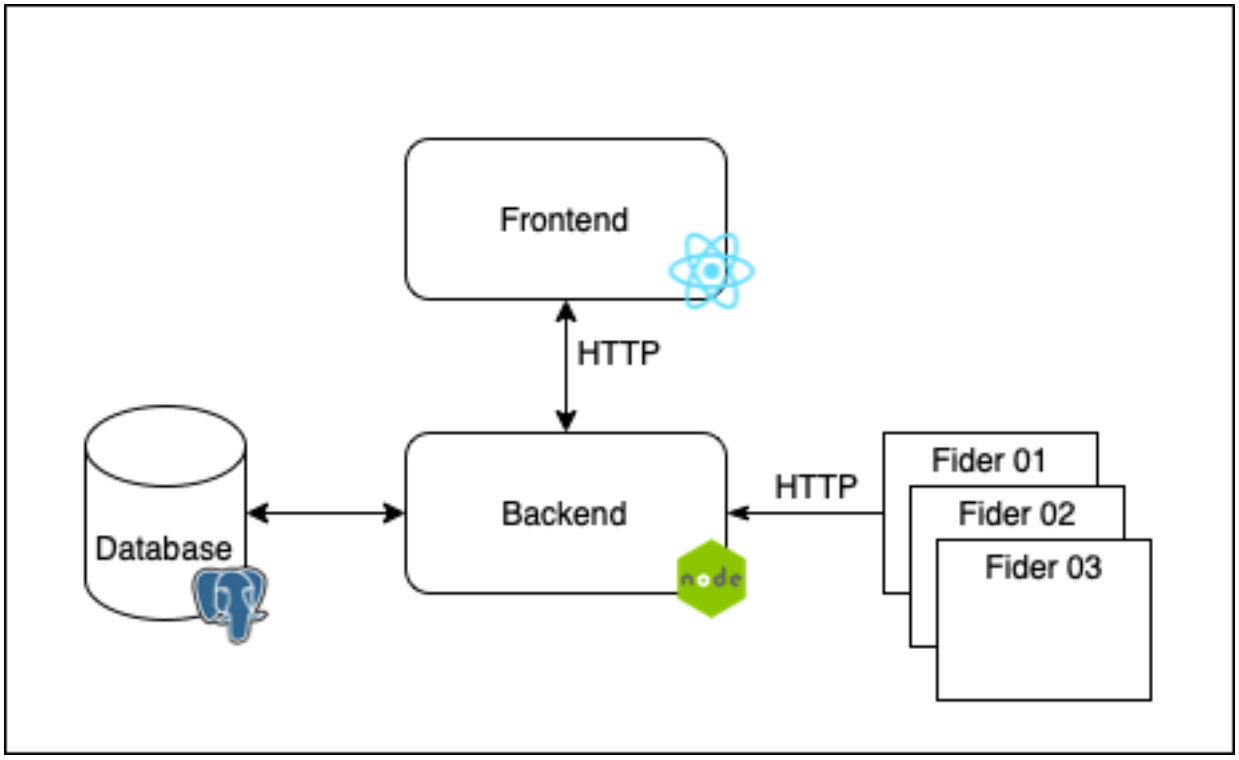
A ideia inicial para o desenvolvimento do *backend* era utilizar Goji, um framework da linguagem GoLang, como o Fider. Porém, dada a restrição de tempo e o trabalho que seria empenhado em aprender uma nova linguagem e framework, optou-se por utilizar NodeJS, um framework de Javascript.

O banco de dados é o PostgreSQL, assim como no Fider.

6.7 Arquitetura

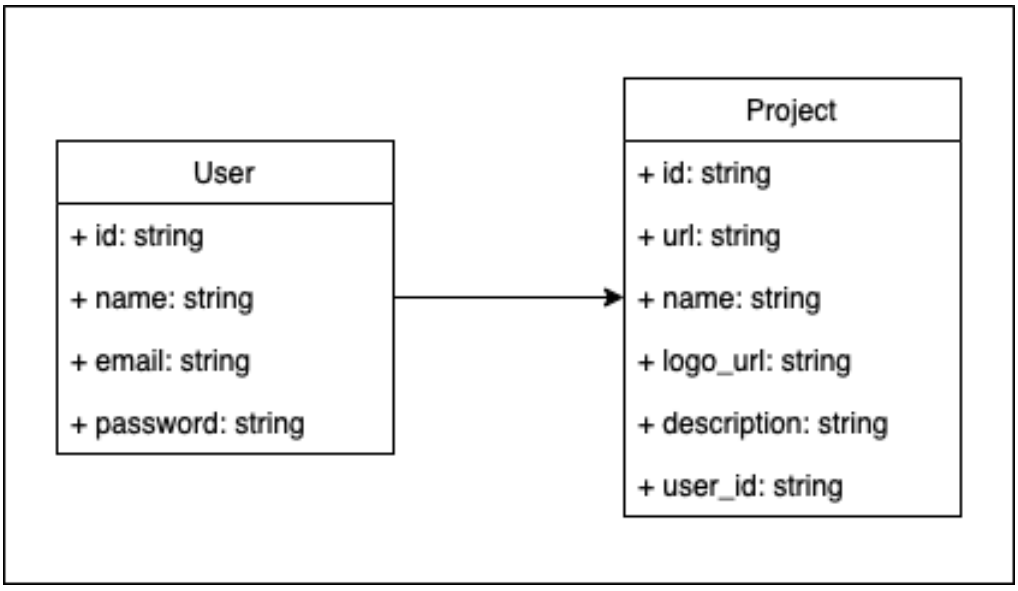
O projeto possui uma arquitetura bem simples, como pode ser observado abaixo:

Figura 1 – Diagrama de Arquitetura



O modelo de dados é básico, pois como todas as informações das solicitações vem diretamente das instâncias do Fider, é necessário gravar apenas os usuários e projetos.

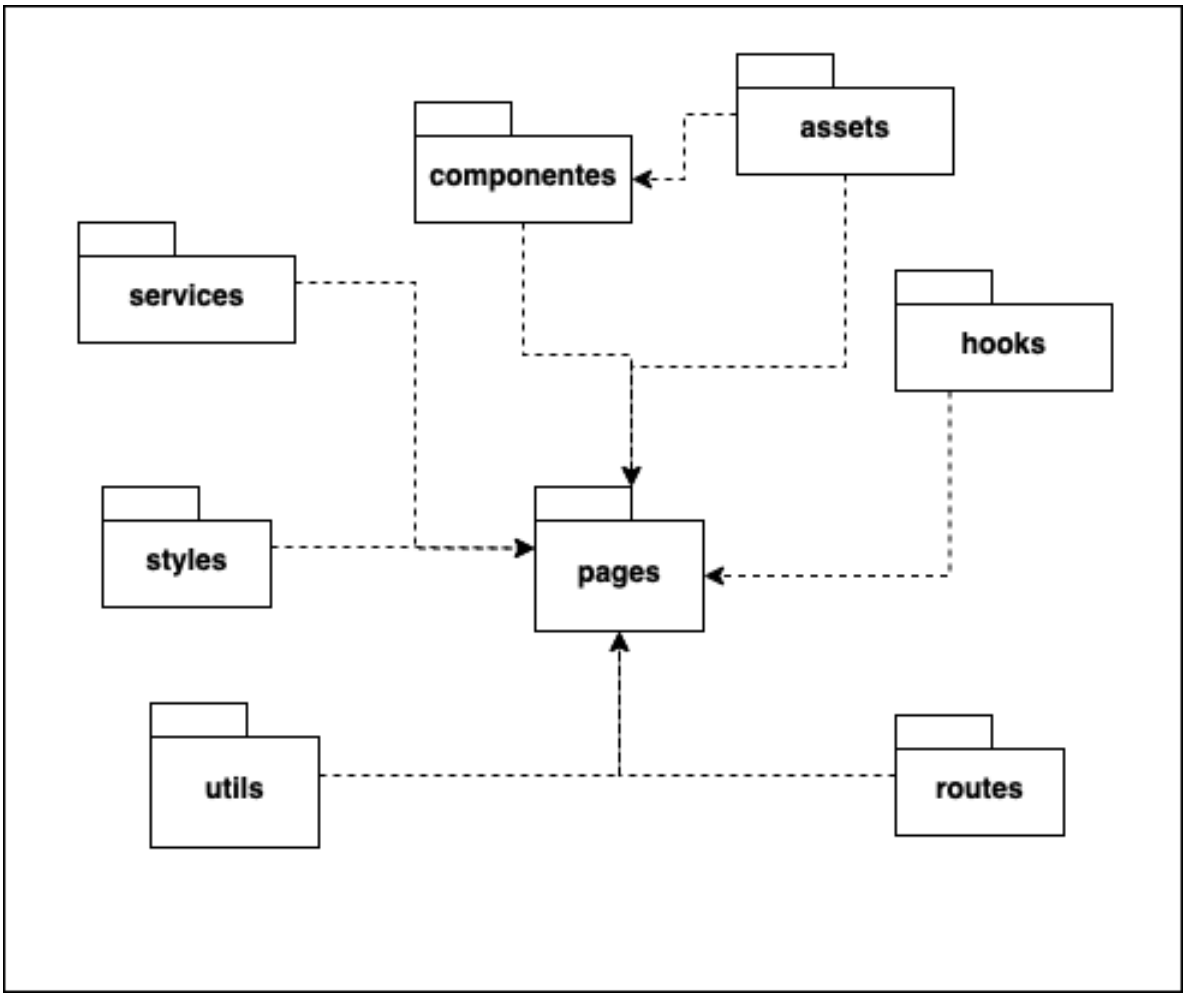
Figura 2 – Modelo de Dados



Abaixo temos o pacotes de dados do *Frontend*, a estrutura usada segue o padrão recomendado pela documentação, alguns pontos interessantes de se ressaltar são:

- O pacote *routes* define o roteamento das páginas, que por sua vez apresentam informações e componentes ao usuário. Para isso é utilizada a biblioteca react-router-dom.
- O *services* é responsável por realizar as requisições para o *backend*, para isso é utilizada a biblioteca axios.
- Os *hooks* são agrupam funcionalidade transversais, que permeiam todo o site. Neste projeto utilizamos para assegurar a autenticação em certas páginas e também aviso de notificações.

Figura 3 – Diagrama de de Pacotes - *Frontend*

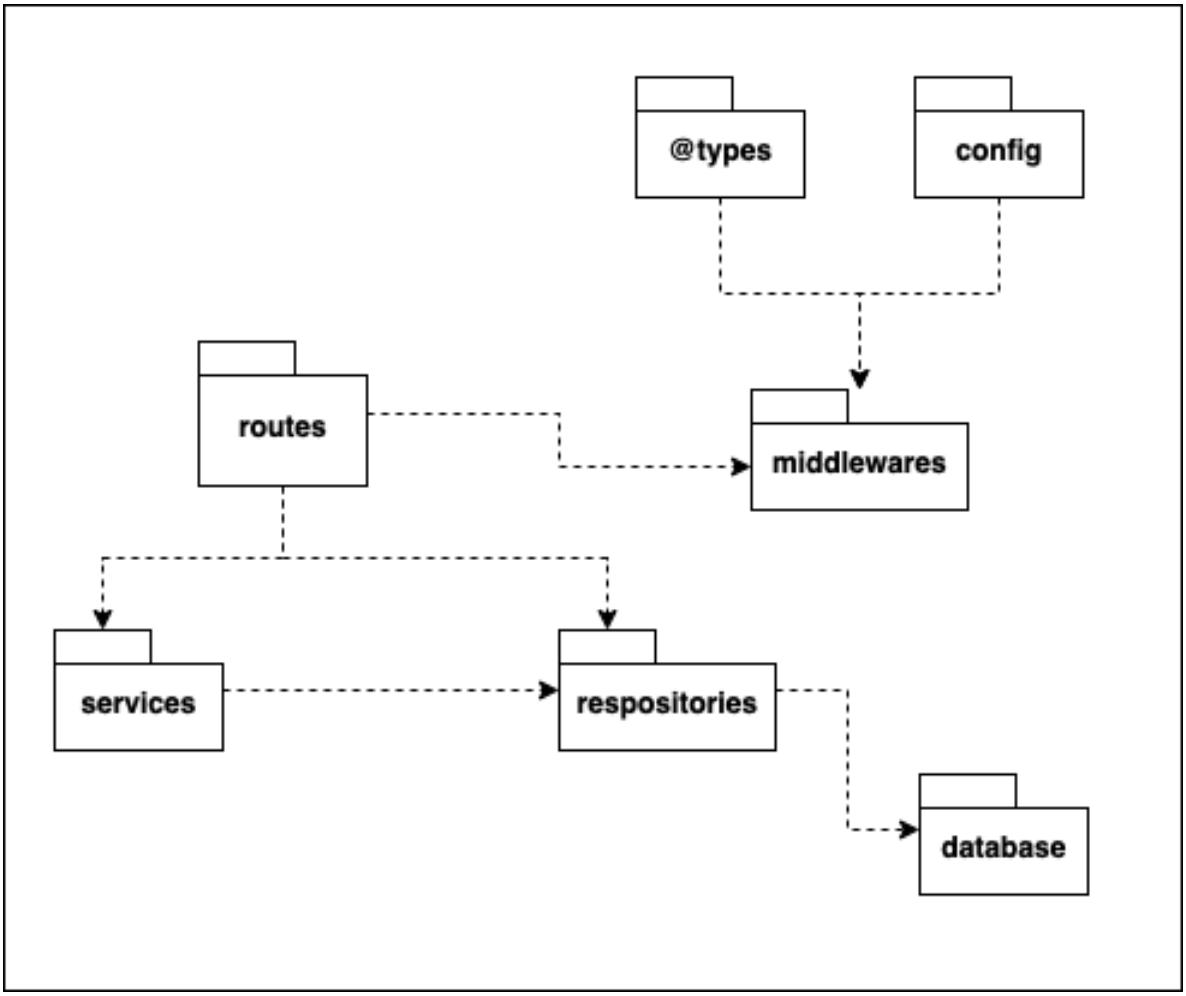


Abaixo temos o pacotes de dados do *Backend*, a estrutura usada segue o padrão recomendado pela documentação. É importante ressaltar alguns pontos que demonstram a preocupação com a qualidade do código, seguindo os princípios SOLID:

- O pacote *routes* agrupa as rotas do projeto, e é reponsável pela recepção e transferência de informações.

- Os *middlewares* são utilizados para envelopar funcionalidades que serão aproveitas em diferentes rotas, funcionando como um *decorator*. No projeto é utilizado para assegurar que certos *endpoints* sejam acessados apenas por usuários autenticados.
- Os *services* possuem as regras de negócio dos módulos do projeto.
- Os *respositories* são abstrações de funcionalidades do banco de dados, é uma camada que complementa o ORM (*Object-Relational Mapping*) utilizado, TypeORM. A sua principal funcionalidade é desacoplar e isolar o banco de outras partes do código, facilitando a manutenção e o *debug*. A implementação foi utilizada focando no *Open-Closed Principle*.

Figura 4 – Diagrama Pacotes - *Backend*



6.8 Resultados

Ao final, foram implementadas 4 das 6 funcionalidades propostas. Mesmo ainda não finalizado, o FC traz um processo de solicitação de funcionalidades mais amigável que

os utilizados atualmente pelas comunidades Debian e KDE. Abaixo encontram-se imagens do projeto.

Figura 5 – Tela de Login

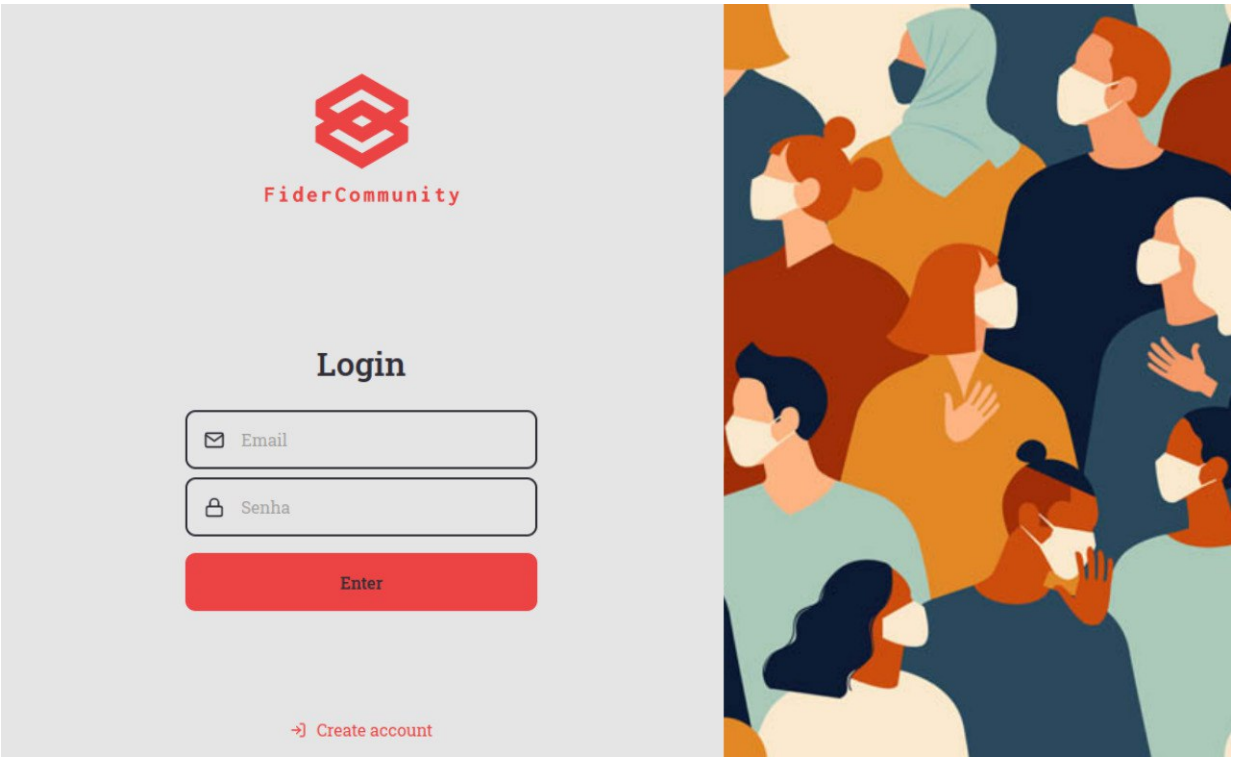


Figura 6 – *Dashboard* de Projetos

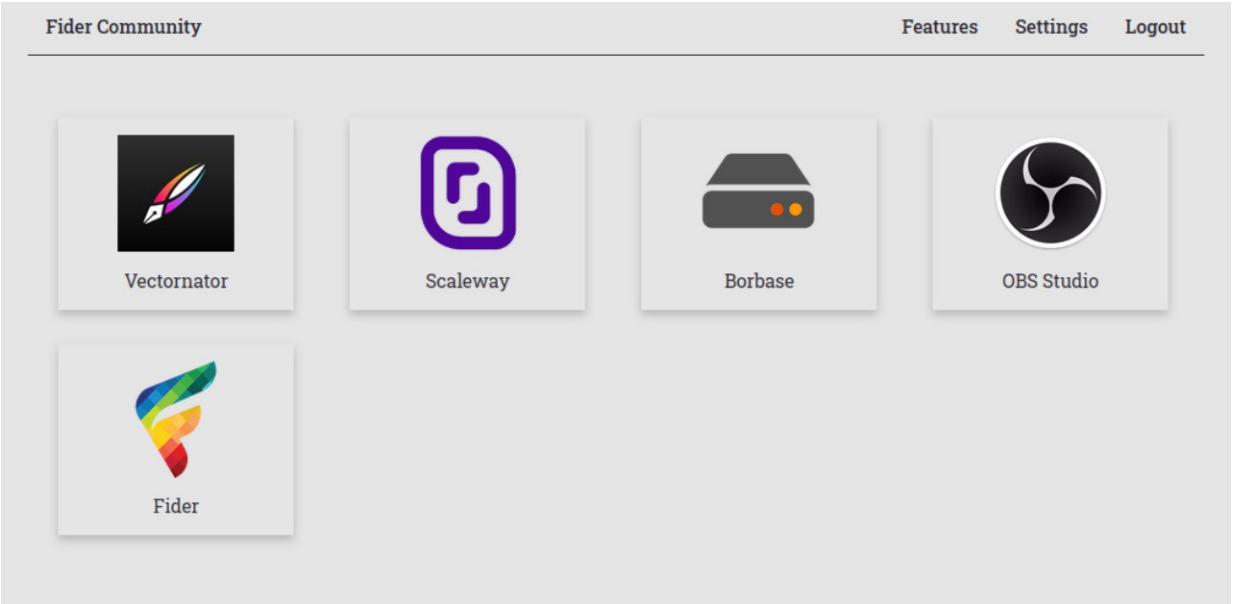


Figura 7 – Tela de Solicitações de Funcionalidades

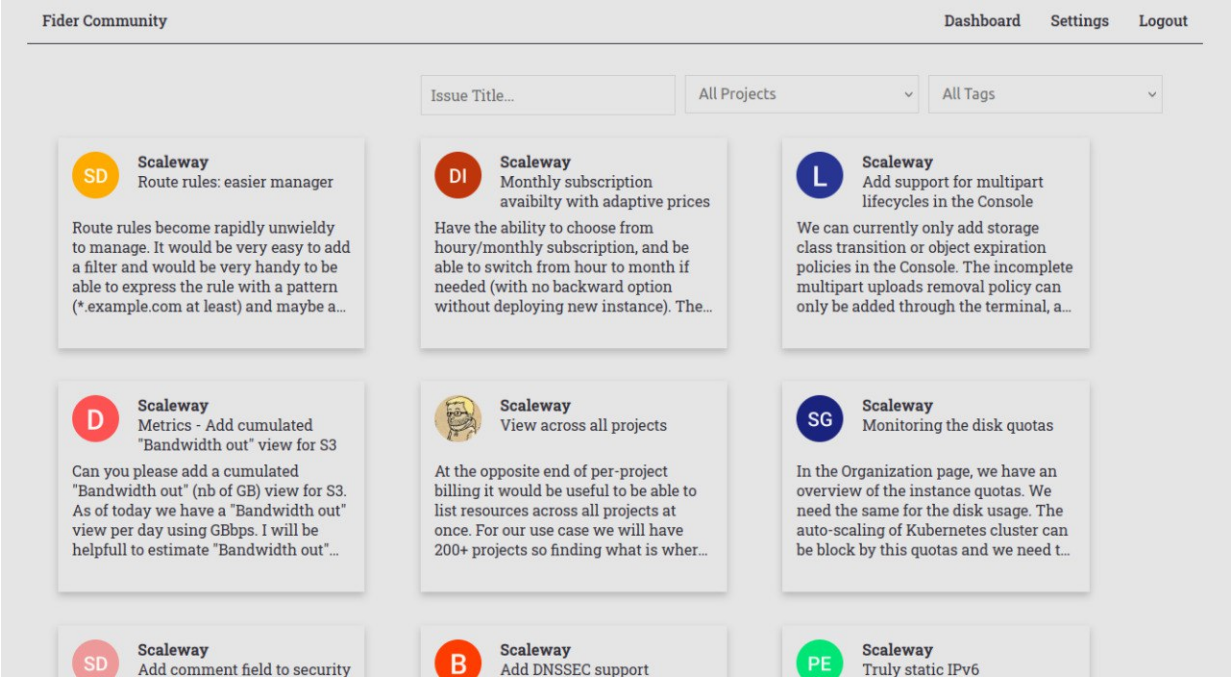
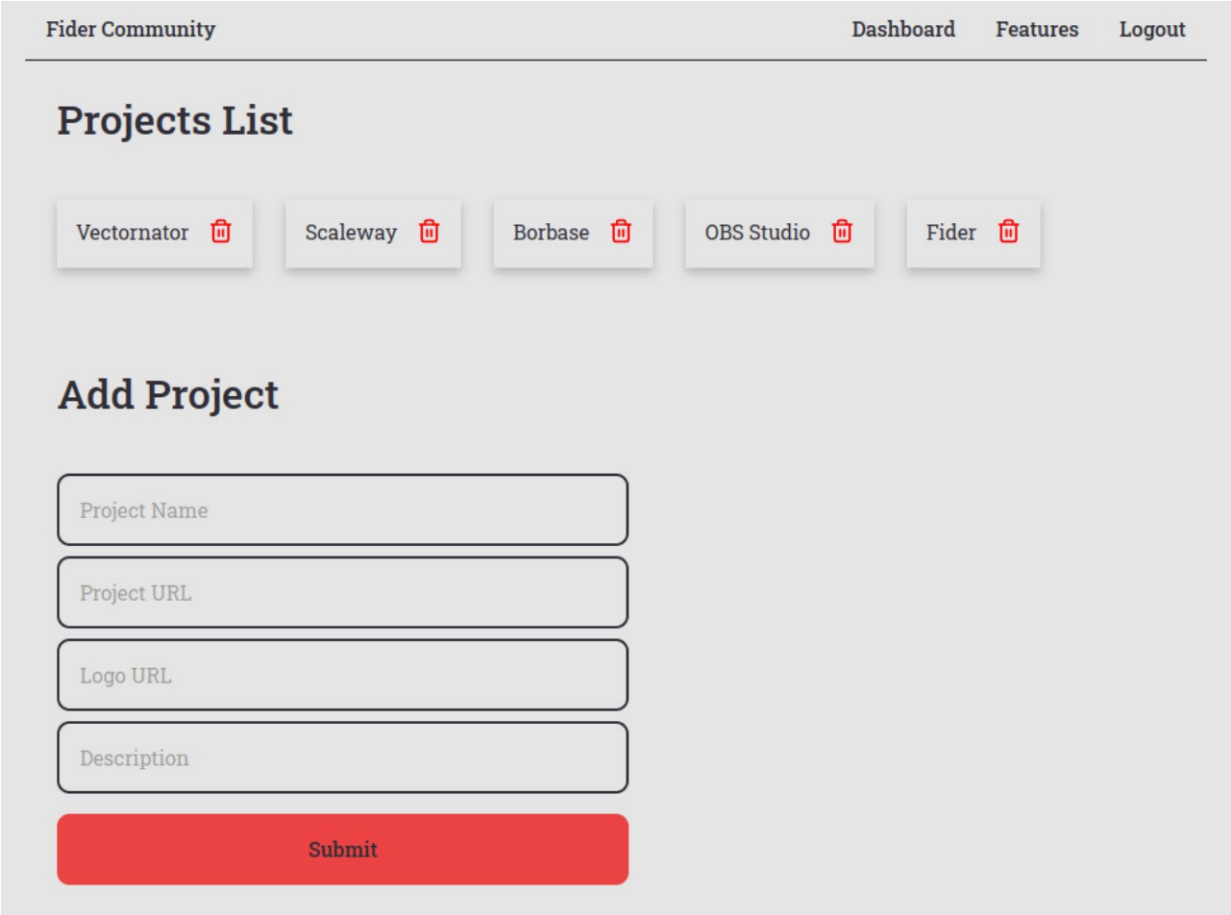


Figura 8 – Tela de Configuração



7 Considerações Finais

Ao fim deste trabalho, fica evidente a relevância do tema de gestão de solicitação de funcionalidades em projetos de software livre. A análise da literatura mostra que este é um problema persistente, que permeia diversas comunidades, não importando seu tamanho e quão bem estão estabelecidas.

Mesmo que não seja um problema presente em todas, ainda sim foi possível identificar vários obstáculos no processo e nas ferramentas utilizadas atualmente. Afinal, como as comunidades possuem diferentes realidades e estágios de maturidade, é previsível que uma única ferramenta não seja adequada a todas.

Como foi apresentado nas sessões de resultados, ainda há várias melhorias a serem feitas. Porém o Fider Community consegue cumprir o que propõe: centralizar e agrupar diferentes instância do Fider para facilitar o acesso.

Para o seu uso por comunidade que já utilizam ferramentas semelhantes, como KDE e Debian, será necessário possibilitar a transferência do histórico.

O FC poderia ser usado por comunidades que estão procurando estruturar seu processo de gestão de solicitações no momento. Mas antes é recomendado a implementação de um processo de autenticação unificado, que melhoria consideravelmente a experiência dos usuários.

É importante ressaltar que não há garantias que o projeto desenvolvido será adotado, afinal para que uma mudança em um processo já consolidado ocorra em centenas de projetos seria necessário uma grande aderência das comunidades.

No entanto, isto não diminui o fato de que é necessário analisar melhor as experiências dos usuários de softwares livres e os proporcionar a melhor experiência possível em todos os aspectos do processo de contribuição.

Não existe, atualmente, uma previsão para o termino de implementação da plataforma, que será deixado para trabalhos futuros.

Apêndices

1 Roteiro Semi-Aberto de Entrevistas

1. Como é feita a comunicação com a comunidade?
2. Como é feita a coleta de feedback de pessoas da comunidade: Como elas solicitam novas features, relatam bugs, como os usuários indicam a satisfação com o produto?
3. Como os mantenedores lidam com essas solicitações: Existe revisão periódica? Essas solicitações são adicionadas a um roadmap?
4. Há a preocupação para colher feedback dos usuários finais? Já foram adotadas ferramentas ou práticas com esse intuito?
5. Vocês enxergam algum problema no modelo adotado atualmente?
6. Quais a soluções vocês acreditam ser boas para superar essa dificuldade?

2 Entrevistas

Abaixo serão apresentadas as respostas dos entrevistados, numeradas de acordo com as perguntas do roteiro acima.

2.1 Debian

Entrevistado(a): Sergio Durigan

Cargo/Competência: Debian Developer

1. Como é feita a comunicação com a comunidade?

   A comunicação é feita, oficialmente, por três canais: o Bug Tracking System (BTS), listas de discussão por e-mail e o IRC.

2. Como é feita a coleta de feedback de pessoas da comunidade: Como elas solicitam novas features, relatam bugs, como os usuários indicam a satisfação com o produto?

   O sistema principal utilizado é o BTS, que é utilizado tanto para registro de bugs como solicitação de features.

   Existem pessoas que também cadastram solicitações de novas features na nossa instância do Gitlab, e mesmo não sendo recomendando há vários desenvolvedores que aceitam.

   Não existe como os usuários indicarem a satisfação, o máximo que temos é um sistema chamado PopCon que vem instalado no Debian e coleta, anonimamente, dados de uso de pacotes. Ele é *OptIn*, ou seja, o usuário tem que habilitá-lo para

que a coleta seja feita, então é provável que ele colete dados de apenas uma pequena minoria. Sendo assim temos informações sobre o uso, mas não sobre o que os usuários acham de pacotes ou suas funcionalidades.

3. Como os mantenedores lidam com essas solicitações: Existe revisão periódica? Essas solicitações são adicionadas a um roadmap?

   O Debian em si não possui um RoadMap. O nosso desenvolvimento é muito distribuído, então o máximo que fazemos é juntarmos-nos em times para trabalharmos em um conjunto de demandas específicas. Adotamos uma postura em que o mantenedor do pacote decide o que é será ou não adicionado, afinal o trabalho é voluntário e cada um decide o que será melhor para seu projeto.

4. Há a preocupação para colher feedback dos usuários finais? Já foram adotadas ferramentas ou práticas com esse intuito?

   Não há um movimento para entrar em contato direto com os usuários finais, mas os canais são abertos para que eles se comuniquem com a comunidade.

5. Vocês enxergam algum problema no modelo adotado atualmente?

   O Debian é um projeto um pouco resiste à mudanças, o que é uma faca de dois gumes: ao mesmo tempo que é uma distribuição muito confiável e segura, onde as mudanças são feitas com muito cuidado, a comunidade não é muito aberta à mudanças.

   O BTS é um software que foi desenvolvido por um dos líderes do projeto Debian em 1994 e sofreu pouquíssimas alterações até hoje. Ele é similar ao Bugzilla, sendo um pouco peculiar: funciona basicamente por e-mail. Mesmo tendo uma interface web, ela é apenas para leitura, não sendo possível manipular qualquer informação (abrir bugs, fechá-los, adicionar tags, etc). Para a manipulação de qualquer informação é necessário utilizar e-mails.

   Eu, e a comunidade Debian no geral, gostamos muitos de como é feita a interação com o BTS. Temos várias funcionalidade que em outros sistemas não é possível de fazer facilmente (principalmente relacionados à pesquisa e geração de relatórios). Mas isso vem com um preço, que no caso eu acho muito alto: acabamos afastando muitos usuários.

   A dificuldade que impomos aos usuários para realizar ações simples é muito alta. Para reportar um novo *bug*, por exemplo, o usuário tem que enviar um e-mail para o endereço do BTS com as tags específicas das informações necessárias para cadastrá-lo (id do pacote, versão, tags do *bug*, etc). Não é um processo simples, para um usuário que não conhece muito do software pode ser difícil. Claro que temos *wikis*

e páginas para ajudar nesse processo, mas é provável que caso um usuário possua algum problema, ele não vai querer perder duas horas aprendendo como reportá-lo. Deveríamos fazer a vida dos usuários mais fácil possível.

Também há as listas de discussão, que também são por e-mail, as quais eu particularmente gosto muito e estou cadastrado em dezenas. Nelas já não vejo que o problema é a dificuldade de acessar ou acompanhar, mas que não atrai as pessoas novas, sendo usada apenas pela comunidade que já está acostumada. Para mim, um projeto que é um bom exemplo do que poderia ser feito é o Fedora: eles possuem as listas de discussão por e-mail, que também estão cadastrada no Discourse, um software mais amigável. E isso é o que as pessoas utilizam hoje, não o Discourse em si mas este tipo de tecnologia.

Inclusive existem pessoas no Debian que estão dispostas a configurar um canal do Discourse para o Debian, o qual eu, particularmente, não usaria, mas seria uma ótima ideia para se aproximas da comunidade.

O canal oficial de comunicação hoje é o IRC (Internet Relay Chat), o qual eu particularmente gosto também, mas entendo que a demanda, hoje, é diferente. O fato do Debian utilizar apenas o IRC como meio de comunicação oficial faz com que os usuários finais não interajam muito, e nisso ocorre uma quebra: no IRC temos a comunidade oficial, porém a maior parte dos usuários está no Telegram, que virou o principal canal de comunicação não oficial (ainda mais no Brasil).

Uma parte do motivo de ainda estamos utilizando estas tecnologias dos anos 90 é porque somos, realmente, um projeto resistente à mudanças, mas também por sermos muito cuidados com o que usamos: somos preocupados em usar tecnologias livres, sempre.

Eu, assim como a comunidade, não gostaríamos que que o projeto começasse a usar o Github, por exemplo, por ser proprietário e incentivar práticas proprietárias de código. Seria totalmente contra intuitivo.

Há vários fatores que nos levam a movimentar-nos lentamente, mas olhando para eles vejo um respeito muito grande à liberdade de software e do usuário. O que nos leva a ponderar muito quais tecnologias vão ser aceitar ou não.

6. Quais a soluções vocês acreditam ser boas para superar essa dificuldade?

   Existem pessoas que consideram uma possível solução implementar uma interface web do BTS, eu acredito que já existam implementações boas para nossos problemas. Caso usássemos o Bugzilla ou o GitLab, por exemplo, já seria um grande passo para nos aproximarmos dos usuários finais.

Porém temos o problema do histórico, sendo o BTS um sistema tão antigo, de 1994, temos mais de um milhão de bugs cadastrados com todo o histórico. Mas acredito que o caminho para frente seria realmente transicionar para outra ferramenta.

2.2 Gnome

Entrevistado(a): Georges Gasile Stravacas

Cargo/Competência: Contribuidor e Mantenedor do projeto Gnome

1. Como é feita a comunicação com a comunidade?

   A comunicação em tempo real é feita por três plataformas: o IRC, que é o canal mais antigo, o Matrix, que é mais novo (porém parte da comunidade não adotou) e o Telegram. O Matrix possui uma ponte de comunicação com o IRC, então os usuários que usam apenas uma destas duas plataformas consegue trocar informações sem problemas. Também utilizamos o Discourse, e nele recomendamos que sejam discutidas coisas menos relacionadas ao trabalho do dia-a-dia.

   Há o canal Rocket.Chat, criado apenas para os casos onde é necessário sigilo da informação discutida pelos membros da Fundação Gnome (em casos jurídicos ou discussão de patente, por exemplo), mas não é voltado para a comunidade.

   Além disso também existe o Gitlab, que é onde ocorre a comunicação assíncrona. Ele é utilizado para a comunicação de todas as áreas: marketing, design, programação, etc.

2. Como é feita a coleta de feedback de pessoas da comunidade: Como elas solicitam novas features, relatam bugs, como os usuários indicam a satisfação com o produto?

   Não existe um canal oficial ou forma definida para que as pessoas da comunidade relatem bugs ou solicitem novas features, isso é feito por todos os canais.

3. Como os mantenedores lidam com essas solicitações: Existe revisão periódica? Essas solicitações são adicionadas a um roadmap?

   Vai completamente à gosto dos mantenedores dos projetos. Geralmente quando as pessoas olham de fora, elas veem a comunidade como um bloco unificado de pessoas que seguem os mesmos protocolos. Mas olhando por dentro é muito mais individualizado, temos um conjunto básico de valores que compartilhamos, mas não existe um conjunto de regras como cada projeto ou mantenedor deve seguir.

4. Há a preocupação para colher feedback dos usuários finais? Já foram adotadas ferramentas ou práticas com esse intuito?

Essas perguntas interpretam o projeto Gnome como uma empresa e o projeto como um produto; que faz pesquisa de mercado e separa os seus usuário por pessoas que contribuem e pessoas que o usam. Na minha experiência essa interpretação não explica como a comunidade funciona, pois não acho que existam usuários finais: quando uma pessoa utiliza e gosto do programa, ela está fazendo uma contribuição, quando ela corrige um bug, ela está fazendo uma contribuição, quando ela da o feedback, ela está fazendo uma contribuição. É difícil explicar de maneira sucinta, mas essa linha que separava o que era um usuário final ou não, foi borrando e desapareceu.

Na coleta de feedback dos usuários, uma parte muito forte nossa é a realização de testes de usabilidade, o que é uma preocupação que não vejo em muitas outra comunidades. Temos o cuidado de testar os módulos, e assegurar que as pessoas estão conseguindo executar tarefas de uma forma efetiva. Inclusive recentemente foi lançado o Gnome 40, e houveram várias mudanças que foram motivadas por uma série de testes de usabilidade que foram realizados em 2020.

Esta é nossa principal maneira de coletar feedback dos usuários de forma mais estruturada e menos enviesada. Com isso entendemos o que funciona, o que não funciona e o que podemos melhorar.

5. Vocês enxergam algum problema no modelo adotado atualmente?

   Vejo que o nosso problema principal são os meios de comunicação fragmentados, já que possuímos o IRC, Matrix, Rocket.Chat, Discourse e até o Telegram. Com isso temos muitas conversas fragmentadas e perda de informação. De resto estamos satisfeitos.

6. Quais a soluções vocês acreditam ser boas para superar essa dificuldade?

   Há várias pessoas trabalhando em várias frentes para tentar resolver esse problema da comunicação. Recentemente fizeram uma pesquisa com os contribuidores recorrentes da comunidade para saber quais features elas precisam do IRC que não está presente no Matrix, assim pretendemos agradar à todos e migrar de vez.

2.3 Rocket.Chat

Entrevistado(a): Theo Renck

Cargo/Competência: Vice-Presidente de Soluções Corporativas

1. Como é feita a comunicação com a comunidade?

Há várias canais de comunicação, como o próprio Rocket.chat, Gitlab, fórum oficial do projeto e os CSMs (Customer Service Managers).

2. Como é feita a coleta de feedback de pessoas da comunidade: Como elas solicitam novas features, relatam bugs, como os usuários indicam a satisfação com o produto?

   Não tem como falar dessa parte sem falar de segmentação. Pense em uma pirâmide, no topo temos o segmento *Enterprise*, depois Marketing e na base o todo mundo que nunca vamos converter para versão paga, e nessa temos vários segmentos, inclusive entusiastas de *Open Source*.

   As informações chegam por todos os canais já citados, são organizadas e clusterizadas, para que, independente do canal, os dados seja processado de forma a gerar inteligência de negócio para o *Product Manager*. Dessa forma, criamos um ambiente que o canal é indiferente.

3. Como os mantenedores lidam com essas solicitações: Existe revisão periódica? Essas solicitações são adicionadas a um roadmap?

   Hoje nosso Roadmap apresenta claramente o que queremos fazer. Porém a comunicação com a comunidade ainda carece de explicitar o que será feito pelo time core (time da Rocket.Chat) e o que está disponível para ser feito pela comunidade, para que ninguém pise um no pé do outro.

   A triagem do Github é território do pessoal de gestão de comunidade, e mesmo que estejamos trabalhando nisso, ainda há muita coisa parada, afinal é um projeto gigante. Ainda mais que tivemos mais um pico de solicitações por conta do COVID.

   Nem sempre vamos poder processar todos os tickets/issues, mas ao clusterizar conseguimos identificar solicitações frequentes para investigar melhor.

4. Há a preocupação para colher feedback dos usuários finais? Já foram adotadas ferramentas ou práticas com esse intuito?

   Hoje temos NPS (Net Promoter Score) embutido no produto que é norteadora do nosso processo. E dependendo do segmento temos processos diferentes, para clientes *enterprise* temos sessões de *discory* para, não só melhorarmos o produto, mas também para descobrir suas dores.

   Além disso, há o processo onde primeiro fazemos consolidação local para depois fazermos a priorização global, que será mandada para o *Product Manager*.

5. Vocês enxergam algum problema no modelo adotado atualmente

Acreditamos que os processos e ferramentas que utilizamos cobrem nossas necessidades, até porque viemos revisando ao longo do tempo, e o que estou falando vem do nosso ciclo de revisão de processo e aprendizado durante anos.

6. Quais a soluções vocês acreditam ser boas para superar a dificuldade em relação à falta de transparência do Roadmap?

   Em relação ao *Roadmap*, temos de sofisticar essa comunicação, para definir o que queremos que esteja disponível apenas no produto pago, o que queremos que vá para o *marketplace* através de *apps*, etc.

2.4 EOS Design System

Entrevistado(a): Cynthia Sanchez

Cargo/Competência: Gerente de Produto

1. Como é feita a comunicação com a comunidade?

   A comunicação é feita pelo nosso canal no Slack, pelas *issues* do Gitlab e, raramente, pela lista de discussão do e-mail.

2. Como é feita a coleta de feedback de pessoas da comunidade: Como elas solicitam novas features, relatam bugs, como os usuários indicam a satisfação com o produto?

   A ferramenta que usamos para coletar os *feedbacks* dos usuários são as *isseus* do Gitlab, no momento. Por lá entram principalmente requisições de features e ideias de melhorias.

   Temos o canal da comunidade no Slack, e conversando com as pessoas da comunidade muitas vezes surgem ideias novas.

   Algumas vezes, raramente para ser sincera, algumas requisições chegam por lista de e-mail. Mas eu diria que mais de 90% chegam pelas issues mesmo.

3. Como os mantenedores lidam com essas solicitações: Existe revisão periódica? Essas solicitações são adicionadas a um roadmap?

   Eu sou a Gerente de Produtos, então o meu trabalho é coletar as issues e as priorizar, e junto com as diretrizes do projeto planejo o *roadmap*. Acaba que há sugestões que serão puxadas para serem feitas agora, outras que serão deixadas para o futuro e acontece de algumas serem descartadas, caso não seja do interesse do time. Essas *issues* descartadas ficam disponíveis para caso alguém da comunidade queira implementada.

Somos um comunidade não convencional nesse sentido, não é muito comum haver um gerente de produtos. Começamos como um projeto do Suse Linux, onde eu era gerente de projetos, e decidimos nos tornar *Open Source*. E continuamos ter um certo controle, onde tomamos decisões quando não há consenso unanime na comunidade.

Temos um plano, um roadmap e objetivos bem claros: de onde vamos, quais problemas queremos resolver e os grupos que queremos focar. Acredito que isso é um dos elementos do nosso sucesso, junto com a comunidade que é muito proativa, que quando tem uma ideia eles vão lá e implementam.

4. Há a preocupação para colher feedback dos usuários finais? Já foram adotadas ferramentas ou práticas com esse intuito?

   Nesse sentido não temos um processo montado, faltam recursos, principalmente de pessoas, para poder melhorar a comunicação com a comunidade. Em geral os usuários que fazem o contato.

5. Vocês enxergam algum problema no modelo adotado atualmente?

   Temos vários problemas, principalmente causado pela falta de recursos. Mas na questão da comunicação, o problema é não termos visibilidade.

6. Quais a soluções vocês acreditam ser boas para superar essa dificuldade?

   Como não somos uma comunidade com muita visibilidade, então deveríamos documentar mais, criar conteúdos de vídeos mostrando como usar o *framework*, conversar com outras comunidades e mostrar realmente o que fazemos.

   E já que não temos muito engajamento com feedbacks, também não temos tantas requisições de feature. Dá para pensar como um funil, primeiro precisamos que as pessoas nos conheçam, depois usem o *framework* e só depois elas vão colocar os *inputs* para que possamos trabalhar em cima.

2.5 Radar Parlamentar

Entrevistados: Haydee Svab, Diego Oliveira e Leonardo Leite

Cargo/Competência: Fundadores e Mantenedores

1. Como é feita a comunicação com a comunidade?

   Geralmente o primeiro contato é por e-mail, já que usamos lista de discussão para comunicar com os alunos das universidades que contribuem com o projeto. Essa foi nossa tentativa de fazer uma comunidade. As rede sociais do projeto temos agidos muito pouco, principalmente porque nesse último ano de pandemia não tivemos

eventos, que costumávamos nos organizar para partipar, e isso gerava um certo movimento nas redes. Teve, também, pessoas que fomos conhecendo em eventos que apresentamos o projeto, às vezes via rede social (não as do projeto, mas as nossas). Não temos uma estrutura pensada de comunicação, então a demanda da comunidade é muito mais informal do que sistematizada.

2. Como é feita a coleta de feedback de pessoas da comunidade: Como elas solicitam novas features, relatam bugs, como os usuários indicam a satisfação com o produto?

   Como foi dito, não temos uma estrutura definida para comunicação.

3. Como os mantenedores lidam com essas solicitações: Existe revisão periódica? Essas solicitações são adicionadas a um roadmap?

   Nós documentamos as demandas pelo nosso *issue tracker*, o Gitlab, mas essa parte de priorização depende muito do momento.

   Quando fazemos uma parceria com alguma faculdade, como a USP ou a UnB, e nós temos mais pessoas contribuindo, nós priorizamos as issues e alinhamos isso com o interesse dos alunos que vão contribuir, organizando em *sprints*.

   Mas costumamos dizer que as *sprints* são orientadas à *hackatons*, então quando nos encontramos o eventos selecionamos uma feature que caiba dentro desse tempo específico, que estejamos motivados pra fazer e que julgamos que já foi pedida algumas vezes pela comunidade.

4. Há a preocupação para colher feedback dos usuários finais? Já foram adotadas ferramentas ou práticas com esse intuito?

   Em relação à coleta de feedback dos usuários, nós não temos força de trabalho para isso. Inclusive estamos aberto caso a sua pesquisa venha com alguma ferramenta ou metodologia que possa ajudar nessa parte.

5. Vocês enxergam algum problema no modelo adotado atualmente?

   O principal problema é a falta de pessoas para melhorar a comunicação e, principalmente, fazer trabalhos operacionais.

   Já deixamos de implementar ideias que poderiam aumentar o engajamento do projeto por falta de alguém que cuide do operacional e da articulação política. Por exemplo, todos os anos na época de eleição para prefeitos e governadores nos pedem um ranking de similaridade para saber com qual o usuário mais se identifica. Mas isso exigiria selecionar uma amostra de proposições legislativas, arbitrar pesos, notas, atualizar essa amostragem e vários outros processo. Esse é um exemplo que

simplesmente falamos que não vamos fazer, já que não temos pessoas fixas para realizar este trabalho.

6. Quais a soluções vocês acreditam ser boas para superar essa dificuldade?

   Dinheiro. Nunca chegamos a aplicar para financiamentos, mas caso conseguíssemos conseguir bolsistas fixos da área de comunicação, ciência política e desenvolvimento já conseguiríamos resolver nosso problema.

2.6 KDE

Entrevistados: Tomaz Canabrava

Cargo/Competência: Mantenedor dos Projetos Plasma e Konsole, integrante do Community Work Group, KDE Network e KDIV.

1. Como é feita a comunicação com a comunidade?

   A gente tem muita comunicação comunitária: grupos oficiais do Redit, no Facebook, no Telegram, mIRC, Matrix e Discord. E é bem fácil ter um grupo oficial do KDE, basta você ser um contribuidor e abrir um grupo.

   Além disso temos wikis, forums e listas de discussões. Não forçamos ninguém a usar uma tecnologia específica, então qualquer pessoa que tenha pelo menos um e-mail consegue entrar em contato com a gente.

2. Como é feita a coleta de feedback de pessoas da comunidade: Como elas solicitam novas features, relatam bugs, como os usuários indicam a satisfação com o produto?

   Usamos ferramentas como o Bugzilla, Phabricator e as *issues* do Gitlab. Além disso, muitas vezes entram pelos canais de comunicação.

   Aceitamos *feature requests* e *bug post* por Redit, Facebook, Telegram e mIRC. Se acharmos a ideia bacana a gente abre para a pessoa no Bugzilla e, caso não tenhamos todas as informações, passamos o link para ela completar.

   Bugs e *feature requests* entram pelo mesmo canal (Bugzilla), o que muda é apenas as tags.

3. Como os mantenedores lidam com essas solicitações: Existe revisão periódica? Essas solicitações são adicionadas a um roadmap?

   Não temos um *roadmap* centralizado, mas como são vários projetos é possível que algum deles tenha.

   Em relação à solicitação: é software livre, o que significa que primeiro vem a sua vida particular, depois o seu trabalho, e quando sobra tempo o software livre. Isso

significa que pode demorar 3 a 4 meses para algo ser implementado, ou algumas coisas não sejam implementadas mesmo sendo bacanas. Essa parte é um pouco chata mas é assim que funciona.

4. Há a preocupação para colher feedback dos usuários finais? Já foram adotadas ferramentas ou práticas com esse intuito?

   Isso é muito difícil de te responder à nível KDE, é quase impossível na verdade. Dentro da comunidade temos projetos muito grandes em que todos os processos definidos são feitos pelos seus desenvolvedores e não necessariamente são iguais a outros.

   Por exemplo o Krita, é um projeto bem grande, com mais de cinco milhões de linhas de código, e eles possuem os seus processos. Não vamos forçar, como entidade KDE, que eles mudem só porque queremos. O KDE é mais um guarda-chuva de projetos: vamos te dar um espaço, apoio e mentoria. Vamos no máximo tentar te encaminhar em processos que já usamos, mas não forçar.

   Com isso posso te falar de um projeto que eu participo: para o Plasma a forma que interagimos são por canais (Telegram, MIRC ou Matrix) e então a sua ideia vai para o Phabricator onde será discutido à nível de software e avaliada se e quando será implementado. Caso seja, quando implementado vamos colocar a pessoa que solicitou dentro do processo de *merge request* para mostrar se o que foi feito atende o que foi pedido.

5. Vocês enxergam algum problema no modelo adotado atualmente?

   Temos problemas com a nossa instância do Bugzilla. Ela é ruim para os padrões de hoje: o site parece, para mim, algo vindo da década de 90. Ele possui grandes problema em relação à usabilidade, um exemplo é nem prover a visualização de imagens, elas só podem ser visualizadas em uma outra aba. O problema é que temos 26 anos de histórico, e não podemos simplesmente ignorar isso. Estamos tentando levar para ferramentas mais novas porém não é simples.

   Já começos a migrar para as issues do Gitlab em alguns projetos, mas com isso temos alguns outros problemas: a ferramenta de busca do Gitlab é muito inferior à do Bugzilla. É difícil achar um sapato que encaixa no pé de todo mundo.

   Não obrigamos os usuários a fazerem *bug report*, se eles conversarem com a gente nós mesmo podemos abrir, porque sabemos que o sistema não é muito bom. E estamos no movendo para tirar o sistema, que era o que tínhamos na época.

6. Quais a soluções vocês acreditam ser boas para superar essa dificuldade?

Já fizemos algumas coisas: caso algum programa do KDE *crashe*, é aberta uma caixa para enviar um *bug report*, então isso já ajuda. Não ajuda no caso de mandar ideia de *feature*, mas temos medo de colocar algum lugar nos aplicativos para isso e sermos imundado por *spam*. Inclusive já tivemos alguns problemas com isso, o IRC mesmo caiu há 2 semanas por isso.

Referências

[1] choosealicense.com. Choose an open source license, 2020. URL https://choosealicense.com/. Citado na página 15.

[2] J. Dinkelacker, P. K. Garg, R. Miller, and D. Nelson. Progressive open source. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, page 177–184, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 158113472X. doi: 10.1145/581339.581363. URL https://doi-org.ez54.periodicos.capes.gov.br/10.1145/581339.581363. Citado na página 15.

[3] O. Gassmann and E. Enkel. Towards a theory of open innovation: three core process archetypes. 2004. Citado na página 16.

[4] R. Glassey. Adopting git/github within teaching: A survey of tool support. In *Proceedings of the ACM Conference on Global Computing Education*, pages 143–149, 2019. Citado na página 12.

[5] P. Heck and A. Zaidman. An analysis of requirements evolution in open source projects: Recommendations for issue trackers. In *Proceedings of the 2013 International workshop on principles of software evolution*, pages 43–52, 2013. Citado 2 vezes nas páginas 16 e 18.

[6] A. K. Kakar. Separating the wheat from the chaff: Extracting business value from feature requests posted in user forums. *Journal of Organizational and End User Computing (JOEUC)*, 28(2):124–141, 2016. Citado na página 15.

[7] J. Longo and T. M. Kelley. Use of github as a platform for open collaboration on text documents. In *Proceedings of the 11th International Symposium on Open Collaboration*, pages 1–2, 2015. Citado na página 12.

[8] W. Maalej and H. Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *2015 IEEE 23rd international requirements engineering conference (RE)*, pages 116–125. IEEE, 2015. Citado na página 18.

[9] W. Maalej, H.-J. Happel, and A. Rashid. When users become collaborators: towards continuous and context-aware user input. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 981–990, 2009. Citado na página 15.

[10] N. McDonald and S. Goggins. Performance and participation in open source software on github. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*,

CHI EA '13, page 139–144, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450319522. doi: 10.1145/2468356.2468382. URL https://doi-org.ez54.periodicos.capes.gov.br/10.1145/2468356.2468382. Citado na página 12.

[11] T. Merten, M. Falis, P. Hübner, T. Quirchmayr, S. Bürsner, and B. Paech. Software feature request detection in issue tracking systems. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 166–175. IEEE, 2016. Citado 2 vezes nas páginas 18 e 19.

[12] opensource.com. What is open source?, 2020. URL https://opensource.com/resources/what-open-source. Citado 2 vezes nas páginas 12 e 15.

[13] M. W. Purcell. Toward understanding new feature request systems as participation architectures for supporting open innovation. In *Proceedings of the 11th International Symposium on Open Collaboration*, pages 1–4, 2015. Citado na página 16.

[14] F. A. Shah, K. Sirts, and D. Pfahl. Using app reviews for competitive analysis: tool support. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*, pages 40–46, 2019. Citado na página 15.