UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

LUCAS VINICIUS RIBEIRO

UM ESTUDO EXPLORATÓRIO SOBRE A ADOÇÃO DE BIBLIOTECAS DE INTERFACE GRÁFICA DE USUÁRIO EM PROJETOS DE SOFTWARE LIVRE

CAMPO MOURÃO

2021

LUCAS VINICIUS RIBEIRO

UM ESTUDO EXPLORATÓRIO SOBRE A ADOÇÃO DE BIBLIOTECAS DE INTERFACE GRÁFICA DE USUÁRIO EM PROJETOS DE SOFTWARE LIVRE

An Exploratory Study on the Adoption of Graphical User Interface Libraries in Open Source Projects

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Igor Scaliante Wiese

CAMPO MOURÃO

2021

LUCAS VINICIUS RIBEIRO

UM ESTUDO EXPLORATÓRIO SOBRE A ADOÇÃO DE BIBLIOTECAS DE INTERFACE GRÁFICA DE USUÁRIO EM PROJETOS DE SOFTWARE LIVRE

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 10/novembro/2021

Igor Scaliante Wiese
Doutorado
UTFPR

André Luiz Satoshi Kawamoto
Doutorado
UTFPR

Ivanilton Polato
Doutorado
UTFPR

CAMPO MOURÃO

2021

AGRADECIMENTOS

O presente trabalho representa, para mim, o encerramento de um ciclo de 9 anos de UTFPR-CM, sendo 4 anos de Curso Técnico Integrado em Informática e 5 anos de Bacharelado em Ciência da Computação. Durante esse período, tive a oportunidade de viver inúmeros momentos especiais e o privilégio de conhecer pessoas incríveis. Por isso, não poderia deixar de registrar a minha gratidão por tudo aquilo que me fez chegar até aqui.

Primeiramente, agradeço a Deus pelo dom da vida, pela saúde e por todas bênçãos derramadas sob o meu caminho durante todos esses anos.

Agradeço aos meus pais, Paulo Sergio Ribeiro e Katia Cristina Rebello Ribeiro por tudo o que fizeram e ainda fazem por mim. Pelo suporte emocional e financeiro, pelo apoio, motivação, carinho, e principalmente, pelo amor incondicional. Sem dúvidas os meus maiores exemplos de vida, perseverança, dedicação e amor. Ainda sobre a família, agradeço ao meu irmão e padrinho Guilherme Fernando Ribeiro, não só por ter sido meu companheiro de moradia durante os meus três primeiros anos em Campo Mourão, mas por todo o cuidado, por todos os conselhos e por ser o meu maior exemplo de dedicação, comprometimento e trabalho duro. Estendo meus agradecimentos à minha cunhada Thays Furlan, aos meus avós Valter e Cecília, aos meus tios, tias e primos que sempre estiveram presentes durante a minha caminhada.

Agradeço também a todos os meus professores da UTFPR com os quais tive a oportunidade de aprender, receber seus ensinamentos, conselhos e apoio. A vocês a minha admiração e gratidão por terem contribuído tanto para a minha formação como pessoa e profissional. Em especial, agradeço ao meu orientador Igor Scaliante Wiese por todo o apoio, orientações e conselhos durante o desenvolvimento do trabalho. Estendo meus agradecimentos aos professores (e amigos) que são mais que especiais para mim: Rodrigo Campiolo, Radames Halmeman, Edson Hirata, Flávia Cardoso, Marli Okada, Devanir Canovas e a minha segunda mãe, Ana Paula Chaves.

Agradeço também ao meu amigo Natan José Rodrigues pelo companheirismo e paciência durante os 5 anos em que dividimos moradia em Campo Mourão. Estendo meus agradecimentos a todos os meus amigos de infância, aos amigos que fiz durante o curso técnico e aos amigos que pude conhecer durante o bacharelado, os quais foram fundamentais e me deram todo o apoio necessário durante a caminhada.

De todo o meu coração, agradeço à Universidade Tecnológica Federal do Paraná - câmpus Campo Mourão, por ter sido a minha segunda casa durante esses 9 anos. A todos professores, servidores, técnicos administrativos, zeladores e diretores que fazem essa instituição ser tão incrível, o meu muito obrigado.

RESUMO

Bibliotecas são componentes de software reutilizáveis desenvolvidos para serem utilizados por qualquer pessoa ou projeto além do desenvolvedor original da biblioteca. Seu uso é importante por contribuir com economia de tempo e esforço durante o processo de desenvolvimento de software. Pesquisas exploram diferentes aspectos relacionados ao uso de bibliotecas de software, tais como, popularidade, riscos envolvidos, processo evolutivo e atualizações, critérios para desenvolvimento de boas bibliotecas, recomendação de *experts*, entre outros. No contexto das interfaces gráficas de usuário, um estudo constatou que 34% das interfaces desenvolvidas são apoiadas por bibliotecas de componentes de interface gráfica de usuário. No entanto, apesar da importância e relevância que bibliotecas e interfaces gráficas de usuário têm nos processos de desenvolvimento de software, não existem estudos, até o presente momento, que exploram este cenário. Portanto, o objetivo desta pesquisa consiste em compreender como ocorre a adoção de bibliotecas de interface gráfica de usuário em projetos de software livre. Para alcançar este objetivo, foram selecionadas as principais bibliotecas de interface de usuário de 6 linguagens de programação, sendo elas C, C++, C#, Java, JavaScript e Python. Assim, foram definidas expressões de busca para tais bibliotecas a fim de identificar seu uso em projetos de software livre do GitHub. Além disso, foram minerados os *commits* de tais projetos a fim de explorar a relação das bibliotecas mais populares com outras partes do código-fonte dos projetos. Os resultados foram analisados estatisticamente, tendo identificado que as bibliotecas mais populares de suas respectivas linguagens foram Win32 (C), Qt (C++), WindowsForms (C#), AWT (Java), React (JavaScript) e PyQt (Python). Constatou-se ainda que boa parte dos *commits* que possuem alterações em arquivos de interface utilizando as bibliotecas mais populares, também possuem alterações em arquivos de outras partes do código-fonte dos projetos. Ainda assim, a maioria dos *commits* observados compreendem alterações somente na interface gráfica de usuário do projeto. Se tratando do uso da biblioteca Win32 nos projetos C, não foi possível verificar se as alterações analisadas compreendem o uso de elementos de interface gráfica de usuário da biblioteca. Dentre os projetos C++, que utilizam a biblioteca Qt, observou-se que a maioria das alterações compreendem mudanças somente na interface gráfica de usuário do projeto. Para os projetos C#, que utilizam a biblioteca WindowsForms, descobriu-se que há baixa relação da biblioteca com outras partes do projeto. Sobre os projetos Java, que utilizam a biblioteca AWT, observou-se que alterações envolvendo a interface de usuário também demandam alterações em outras funcionalidades do projeto. Já se tratando dos projetos JavaScript, utilizando a biblioteca React, os resultados apontam para uma relação entre interfaces de usuário e uso de casos de teste. Por fim, em relação aos projetos Python, que utilizam a biblioteca PyQt, descobriu-se que existe pouca relação entre a biblioteca e arquivos de testes e do *back-end* dos projetos.

Palavras-chaves: Bibliotecas. Software Livre. Interface Gráfica de Usuário.

ABSTRACT

Libraries are reusable software components designed to be used by anyone or any project other than the original developer of the library. Its use is important as it contributes to saving time and effort during the software development process. Research explores different aspects related to the use of software libraries, such as popularity, risks involved, evolutionary process and updates, criteria for developing good libraries, *experts* recommendation, among others. In the context of graphical user interfaces, a study found that 34% of the developed interfaces are supported by graphical user interface component libraries. However, despite the importance and relevance that libraries and graphical user interfaces have in software development processes, there are no studies, so far, that explore this scenario. Therefore, the objective of this research is to understand how occurs the adoption of graphical user interface libraries in open source software projects. To achieve this goal, the main user interface libraries of 6 programming languages were selected, namely C, C++, C#, Java, JavaScript and Python. Thus, regular expressions were defined for such libraries in order to identify their use in GitHub's open source projects. Furthermore, the *commits* of such projects were mined in order to explore the relationship of the most popular libraries with other parts of the project's source code. The results were statistically analyzed, having identified that the most popular libraries of their respective languages were Win32 (C), Qt (C++), WindowsForms (C#), AWT (Java), React (JavaScript) and PyQt (Python). It was also found that most of the commits that have changes in interface files using the most popular libraries, also have changes in files from other parts of the source code of the projects. Still, most of the commits observed have changes only to the project's graphical user interface. Regarding the use of the Win32 library in C projects, it was not possible to verify whether the analyzed changes comprise the use of the library's graphical user interface elements. Among the C++ projects that use the Qt library, it was observed that most of the changes comprise changes only in the project's graphical user interface. For C# projects, which use the WindowsForms library, it was found that there is a low relationship between the library and other parts of the project. About Java projects, which use the AWT library, it was observed that changes involving the user interface also demand changes in other project features. As for JavaScript projects, using the React library, the results point to a relationship between user interfaces and the use of test cases. Finally, regarding Python projects, which use the PyQt library, it was found that there is a low relationship between the test library and files and the back-end of the projects.

Keywords: Libraries. Open Source Software. Graphical User Interface.

SUMÁRIO

1 INTRODUÇÃO

Bibliotecas têm sido cada vez mais importantes no dia a dia de um desenvolvedor de software, por permitirem o reuso de código e contribuírem significativamente no aumento da produtividade (LIM, 1994; RAEMAEKERS et al., 2012a). Sojer e Henkel (2010) constataram que o reuso de código corresponde a um dos principais papéis no desenvolvimento de projetos de software livre. No estudo em questão, desenvolvedores foram entrevistados e relataram que, em média, 30% das funcionalidades implementadas em seus projetos foram baseadas no uso de bibliotecas desenvolvidas por terceiros.

Pesquisadores têm explorado diferentes perspectivas relacionadas ao uso de bibliotecas em projetos de software. Lima e Hora (2020) focaram em investigar características de bibliotecas populares, como tamanho, legibilidade, complexidade, documentação, entre outros aspectos. Um outro estudo concentrou-se em analisar os processos de uso e atualização de bibliotecas em projetos de software livre quantificando o risco em potencial que o uso de bibliotecas desatualizadas podem causar (WANG et al., 2020). Sob uma outra perspectiva, Larios-Vargas et al. (2020) concentraram-se em estabelecer critérios para auxiliar o processo de escolha de bibliotecas em projetos de software. Uma outra pesquisa dedicou-se no desenvolvimento de um modelo para recomendação de *experts* em três *frameworks* populares de linguagem JavaScript, sendo eles React, Node-MongoDB e Socket.IO (MONTANDON et al., 2019).

No contexto das interfaces gráficas de usuário, Myers e Rosson (1992) constataram que, em média, 48% do código-fonte dos programas de computador são voltados à interface de usuário. Além disso, o estudo em questão observou que o tempo gasto com desenvolvimento de interfaces de usuário corresponde à 50% do tempo de desenvolvimento dos projetos, e que 34% dos projetos são realizados utilizando bibliotecas para auxiliar o processo de desenvolvimento das interfaces. Estudos anteriores ressaltam a importância das interfaces gráficas de usuário nos projetos de software (MYERS; ROSSON, 1992; XIE; MEMON, 2006; IIVARI et al., 2008). Pesquisas também analisaram e destacaram a importância e as vantagens de usar ferramentas e bibliotecas que apoiam o desenvolvimento de interfaces gráficas de usuário (MYERS, 1995; MYERS et al., 2000).

Desta forma, fica evidente a importância das interfaces de usuário e das bibliotecas que apoiam seu desenvolvimento. Apesar disso, até o presente momento, não existem na literatura estudos que buscam explorar a utilização de bibliotecas de interface gráfica de usuário em projetos de software livre. Portanto, definiu-se como objetivo desta pesquisa realizar um estudo de caso exploratório para compreender como ocorre a adoção de bibliotecas de interface gráfica de usuário em projetos de software livre.

Para alcançar o objetivo proposto, foram selecionadas as principais bibliotecas de interface gráfica de usuário de 6 linguagens de programação, sendo elas C, C++, C#, Java, JavaScript e Python. A partir de então, foram definidas expressões de busca para identificar o uso de tais bibliotecas em projetos de software livre hospedados no GitHub. As expressões de busca foram executadas nos

códigos-fonte dos projetos de maneira automatizada utilizando uma ferramenta de mineração, e os resultados foram validados manualmente por meio de uma interface web. Desta maneira, os dados foram analisados estatisticamente de modo a compreender como se dá a popularidade das bibliotecas analisadas, e qualitativamente de modo a observar-se como as bibliotecas se relacionam com as demais partes dos códigos-fonte dos projetos. Dentre as descobertas obtidas com a realização deste estudo, observou-se que as bibliotecas mais populares foram Win32 (C), Qt (C++), WindowsForms (C#), AWT (Java), React (JavaScript) e PyQt (Python). Além disso, foi possível observar que aproximadamente 30% dos *commits* de projetos de software livre envolvendo o uso da biblioteca de interface gráfica de usuário, também possuem alterações em outras partes dos projetos, como funcionalidades, arquivos de testes e *back-end*. Por outro lado, cerca de 70% dos *commits* que modificam arquivos contendo a biblioteca, possuem alterações somente nos arquivos da interface de usuário do projeto.

O presente trabalho está estruturado da seguinte forma: no primeiro capítulo a pesquisa foi contextualizada, a justificativa apresentada, o objetivo explicitado e algumas descobertas reveladas. O segundo capítulo contempla as principais noções teóricas e conceitos base para o entendimento do trabalho. No terceiro capítulo apresenta-se os trabalhos relacionados. O método, objetivo e questões de pesquisa são revelados no quarto capítulo. No quinto capítulo são apresentados os resultados, implicações e ameaças à validade deste estudo. O sexto capítulo apresenta as conclusões e, ao final, as referências bibliográficas são apresentadas.

2 REFERENCIAL TEÓRICO

O presente capítulo está estruturado em duas seções e apresenta as principais referências teóricas que servirão de base para o desenvolvimento desta pesquisa. A primeira seção apresenta uma visão geral sobre projetos de software livre e sobre o GitHub, plataforma que permite a hospedagem de projetos desta natureza. Na segunda seção é apresentada uma visão geral sobre o conceito e importância das bibliotecas no desenvolvimento de software. Para composição deste capítulo foram utilizados artigos científicos, presentes na literatura, que tratavam dos assuntos em questão.

2.1 Projeto de software livre

O compartilhamento de código entre programadores é uma atividade que acontece desde os primeiros anos da computação, na década de 1940. Já nas décadas de 50 e 60, surgiram os primeiros softwares proprietários, que consistiam em programas vendidos juntos ao hardware do computador (BESSEN, 2002). Mais tarde, no ano de 1985, o cientista da computação Richard Stallman fundou a *Free Software Foundation* com intuito de arrecadar fundos para o desenvolvimento do sistema operacional GNU (STALLMAN, 2002). A partir de então originou-se o movimento de software livre.

De acordo com Kelty (2001), nas últimas décadas o movimento de software livre se tornou constantemente presente na indústria e tecnologia, gerando conferências, investigações acadêmicas e novos empreendimentos. Ao contrário do software proprietário, que tem sua redistribuição, cópia e modificação restringidos pelos direitos autorais do desenvolvedor, o software livre é caracterizado pela liberdade à execução, distribuição, alteração e cópia do mesmo. Ainda segundo Kelty (2001), os usuários de um software livre possuem 4 tipos de liberdade:

- Liberdade para executar o programa, para qualquer propósito;
- Liberdade para estudar como o software funciona e adaptá-lo às suas necessidades;
- Liberdade para redistribuir cópias de forma gratuita ou paga;
- Liberdade para melhorar o software e redistribuí-lo à comunidade.

Conforme Krogh e Hippel (2006), os projetos de software livre geralmente são criados por um único desenvolvedor ou um grupo de desenvolvedores que desejam criar um programa de computador para atender suas próprias necessidades. Hoje em dia, há um grande número de projetos desenvolvidos para diferentes propósitos, desde sistemas operacionais até jogos eletrônicos (KROGH; HIPPEL, 2006).

Segundo Wu e Lin (2001), qualquer pessoa capaz de escrever código é bem-vinda em um projeto de software livre. De maneira complementar, Scacchi et al. (2006) ressaltam que, além de desenvolvedores, muitos usuários finais também contribuem e participam no processo de desenvolvimento de projetos de software livre provendo *feedback*, reportando erros e problemas e sugerindo melhorias de usabilidade. A colaboração coletiva dos desenvolvedores e usuários resulta

na formação de comunidades que contribuem continuamente aos processos de desenvolvimento do software, correção de problemas, erros ou falhas de segurança e afins. Como resultado, tem-se a produção de softwares robustos e modelos de negócios diversificados (WU; LIN, 2001).

De acordo com Scacchi et al. (2006), as comunidades de software livre geralmente são formadas por desenvolvedores e usuários distribuídos ao redor do planeta. Em alguns projetos, organizações contratam e pagam desenvolvedores para trabalhar nos projetos como parte de seus trabalhos. Entretanto, são mais comuns os casos de desenvolvedores que voluntariam seu tempo, esforço e conhecimento para contribuir em projetos de software livre. Scacchi et al. (2006) constataram que, dentre os motivos que levam os desenvolvedores à contribuírem voluntariamente nos projetos, estão as oportunidades de aprendizados, compartilhar conhecimentos, ser reconhecido pela comunidade e construir relacionamentos com outros desenvolvedores.

Com o crescimento do movimento de software livre, criou-se a necessidade de tecnologias e ferramentas para auxiliar o desenvolvimento e controle dos projetos. Atualmente, a mais expressiva plataforma para hospedagem de código e desenvolvimento colaborativo é o GitHub. Na subseção seguinte, será realizada uma abordagem geral sobre o GitHub e seus principais recursos e funcionalidades.

2.1.1 GitHub

De acordo com a definição de Dabbish et al. (2012), GitHub é um *website* de hospedagem de código baseado no sistema de controle de versão Git[1]. Além das funcionalidades de hospedagem de código e gerenciamento de projetos, esta plataforma fornece um conjunto de ferramentas de "codificação social" que tornam as informações e as atividades de um desenvolvedor visíveis para outros usuários. Em agosto de 2019, o GitHub atingiu a marca de mais de 50 milhões de usuários e 100 milhões de repositórios[2].

Os projetos hospedados no GitHub ficam disponíveis em uma página web exclusiva do projeto denominada *repositório*. Além de hospedar os arquivos do código-fonte, o repositório também armazena o histórico de *commits*, as *issues* e outras informações associadas ao projeto. Um repositório pode ser público, de modo que qualquer usuário pode acessá-lo, ou privado, de maneira que somente usuários autorizados podem acessá-lo.

As ações no código-fonte de um projeto incluem fazer *commit*, *fork*, *merge* e *pull request*. Os proprietários do projeto e usuários com permissões de alteração podem fazer *commits*, que são alterações no código, afetando diretamente o projeto original. Já os usuários sem direitos de realizar *commits* devem fazer um *fork* do projeto, criando uma cópia do código-fonte, na qual é possível realizar alterações livremente. Assim, pode ser submetida uma solicitação de alteração do projeto original por meio de um *pull request*. O proprietário do projeto ou um usuário com permissões de

[1] https://git-scm.com/
[2] https://github.com/about

alteração pode aceitar o *pull request*, fazendo um *merge*, que consiste em integrar as mudanças ao projeto original.

As funcionalidades sociais do GitHub envolvem ações de *star*, *watch* e *follow*. Inspirado pelo botão *curtir* das redes sociais, a ação de dar estrela (*star*) a um repositório é uma forma de manifestar interesse, apoio ou satisfação pelo projeto (BEGEL et al., 2013). Já a função *watch* serve para acompanhar e receber notificações das atividades do projeto, bem como novos *pull requests*, *commits* e *issues*. A função *follow* é uma ação realizada entre usuários, que permite demonstrar interesse por suas atividades.

Também existem as funcionalidades *Project Boards* e *Teams*. *Project Boards* permitem organizar e priorizar as tarefas por meio de cartões que contém informações sobre *issues* e *pull requests*. Desta maneira, os cartões podem ser organizados em colunas que indicam o estágio de desenvolvimento em que a tarefa se encontra. Já a funcionalidade de *Teams*, permite criar times de desenvolvedores e atribuir a eles níveis de permissões diferentes, e acesso à *Project Boards* específicas.

O GitHub ainda disponibiliza uma API GraphQL[3] com intuito de prover um grande volume de informações dos repositórios para desenvolvedores que desejam fazer integrações à outros softwares, bem como para que a comunidade científica possa desfrutar destas informações para realizar estudos na área. A API do GitHub será utilizada em uma das etapas do método desta pesquisa para coletar os projetos a serem analisados e suas respectivas informações.

2.2 Biblioteca

O conceito de reutilização de componentes de software foi introduzido na literatura no ano de 1968 por McIlroy et al. (1968). Desde então, pesquisas e estudos vêm sendo realizados neste contexto, por exemplo (MILI et al., 1998; HAEFLIGER et al., 2008; MILEVA et al., 2009).

De acordo com Haefliger et al. (2008), as atividades de reutilização de software podem ser categorizadas em dois principais tipos: *White-box Reuse*, ou reutilização caixa branca, que consiste em incorporar aos arquivos de um software, códigos-fonte externos desenvolvidos por terceiros. Já o segundo tipo, denominado *Black-box Reuse*, ou reutilização caixa preta, significa utilizar bibliotecas de software onde o código-fonte não é visível e nem modificável.

Mais tarde, Raemaekers et al. (2012b) definiram biblioteca de software como um componente de software reutilizável e desenvolvido para ser usado por qualquer pessoa ou entidade além do desenvolvedor original da biblioteca. O uso de bibliotecas em projetos de software contribui para economizar tempo e esforço durante o desenvolvimento, por não precisar reconstruir funcionalidades e componentes já existentes (LIM, 1994; MORISIO et al., 2002b; BALDASSARRE et al., 2005; RAEMAEKERS et al., 2012a). Desta forma, a produtividade dos desenvolvedores é melhorada e os custos para desenvolvimento do software são reduzidos (MORISIO et al., 2002a; CONSTANTINOU et al., 2014; WANG et al., 2020).

[3] https://docs.github.com/en/free-pro-team@latest/graphql

Assim como em muitos outros contextos, no âmbito das bibliotecas em projetos de software também existem desvantagens e riscos relacionadas ao seu uso. Uma das principais desvantagens está relacionada aos custos de manutenção associados ao processo de manter as bibliotecas atualizadas (WANG et al., 2020). Além disso, manter o uso de versões desatualizadas das bibliotecas pode causar *bugs* e falhas de segurança no software. Em contrapartida, novas versões das bibliotecas podem conter mudanças que implicam na refatoração do código-fonte do programa (KIM et al., 2011; BOGART et al., 2016).

No contexto de projetos de software livre, Sojer e Henkel (2010) constataram que, em média, 30% das funcionalidades implementadas nos projetos são baseadas em componentes reutilizados de bibliotecas externas. No presente trabalho, será estudado o uso de bibliotecas de interface gráfica de usuário em projetos de software livre.

3 TRABALHOS RELACIONADOS

Neste capítulo são apresentados trabalhos presentes na literatura que possuem intersecções com o tema central desta pesquisa.

Existem na literatura muitos estudos com foco em dar suporte aos desenvolvedores no processo de desenvolvimento de bibliotecas. Robillard (2009) investigou os aspectos que dificultam o processo de aprendizado do uso de uma biblioteca. Neste mesmo estudo, o autor utilizou questionários e entrevistas com desenvolvedores da Microsoft, e a partir disso constatou que, por exemplo, conteúdos escassos nas documentações é um fator crítico no aprendizado das bibliotecas. Bloch (2006) apresentou alguns critérios associados ao desenvolvimento de uma boa biblioteca, como por exemplo, boa documentação, códigos de exemplo relevantes, métodos e classes com nomes significantes, entre outros. Stylos e Myers (2007) mostraram que bibliotecas bem desenvolvidas são fáceis de usar, além de melhorar a produtividade dos desenvolvedores, prevenir erros e ter poucas chances gerar problemas.

Outros estudos existentes na literatura possuem como foco investigar o processo de evolução das bibliotecas. Xavier et al. (2017) investigaram 317 bibliotecas e os resultados apontaram que 28% das atualizações dessas bibliotecas causam problemas de incompatibilidade com versões anteriores. Brito et al. (2018) constaram que geralmente os problemas causados pelas bibliotecas decorrem de fatores como implementação de novas funcionalidades, melhorias na biblioteca, correção de problemas, entre outros. Hora et al. (2018) realizaram um estudo sobre as mudanças de 118 bibliotecas ao longo do tempo, e constatou-se que 61% dos softwares analisados foram afetados por essas mudanças.

Mais relacionados a este trabalho encontram-se os estudos sobre a popularidade das bibliotecas em repositórios de software. Mileva et al. (2010) analisaram 450 bibliotecas e identificaram quais foram as mais utilizadas ao longo dos anos. Raemaekers et al. (2012b) investigaram a popularidade de bibliotecas em projetos de software e avaliaram os riscos que os projetos possuem de acordo com seu grau de dependência pelas bibliotecas. Já Lima e Hora (2020) concentraram-se em investigar características de bibliotecas populares, como tamanho, legibilidade, complexidade, documentação, entre outros aspectos. Wang et al. (2020) analisaram os riscos em potencial que o uso de bibliotecas desatualizadas podem causar aos projetos de software livre.

É possível observar, portanto, que nenhum dos estudos anteriores investigou a adoção de bibliotecas de interface gráfica de usuário, especificamente. Além disso, as pesquisas apresentadas, em sua grande maioria, possuem como foco estudar muitas bibliotecas de uma única linguagem de programação. No escopo deste estudo, serão investigadas as principais bibliotecas de interface gráfica de usuário de 6 linguagens de programação diferentes. Além disso, o presente estudo concentra-se em analisar a popularidade das bibliotecas estudadas e explorar como as bibliotecas de interface gráfica de usuário mais populares se relacionam com outras partes do código-fonte dos projetos de software-livre.

4 METODOLOGIA

Inicialmente, este capítulo apresenta o objetivo do presente trabalho. Em seguida, são explicitadas as duas questões de pesquisa a serem respondidas. Na sequência, apresenta-se os detalhes da ferramenta de mineração que foi utilizada para coletar e filtrar os projetos, e buscar as expressões de busca das bibliotecas nos repositórios. Por fim, são detalhadas as etapas do método de pesquisa.

4.1 Objetivo

Considerando-se o estado da arte e a lacuna existente na literatura apresentada inicialmente, definiu-se como objetivo desta pesquisa realizar um estudo exploratório para compreender como ocorre a adoção de bibliotecas de interface gráfica de usuário em projetos de software livre. Nesse sentido, analisou-se a popularidade das bibliotecas nos projetos, buscando identificar padrões de alterações e como as bibliotecas de interface gráfica de usuário mais populares se relacionam com outras partes dos projetos em que são utilizadas. Para guiar os passos deste estudo a fim de alcançar o objetivo proposto, foram definidas duas questões de pesquisa que são reveladas na seção seguinte.

> ***Objetivo:*** *Compreender como ocorre a adoção de bibliotecas de interface gráfica de usuário em projetos de software livre.*

4.2 Questões de Pesquisa

Para alcançar o objetivo proposto neste trabalho, duas questões de pesquisa foram definidas, as quais são reveladas e detalhadas a seguir.

QP1: Quais são as bibliotecas de interface gráfica de usuário mais populares entre as 6 linguagens de programação analisadas? Com esta questão de pesquisa, buscou-se identificar a popularidade das bibliotecas nos projetos considerados no escopo deste trabalho. Para isso, realizou-se uma análise estatística da quantidade de projetos em que as bibliotecas são utilizadas, considerando-se projetos e bibliotecas de mesma linguagem. Também analisou-se estatisticamente a quantidade de importações de cada biblioteca dentre os arquivos do código-fonte dos projetos, de modo a identificar a quantidade de arquivos que importam uma biblioteca dentro de um projeto.

QP2: Como as bibliotecas se relacionam com as demais partes do código-fonte dos projetos? Com esta questão, objetivou-se identificar padrões de utilização das bibliotecas com outras partes do código-fonte dos projetos que as utilizam. Para esse fim, analisou-se manualmente *commits* dos projetos, categorizando-os de acordo com aspectos como quantidade de arquivos alterados, mensagem de *commit* e alterações em arquivos de outras partes do projeto que não sejam relacionadas à interface gráfica usuário. Dessa forma, é proposta uma análise qualitativa com intuito de explorar as

mudanças e os padrões de alterações observados. Para isso, foram selecionados somente os projetos que utilizam a biblioteca mais popular de sua respectiva linguagem, de acordo com o que foi observado na primeira questão de pesquisa. Assim, foram analisados 50 *commits* por linguagem de programação, distribuídos aleatoriamente entre os projetos, totalizando 300 *commits* analisados.

4.3 Ferramenta de mineração

A ferramenta em questão, denominada *db-mining*[1], trata-se de um programa desenvolvido com intuito de identificar e quantificar o uso de diferentes bancos de dados em projetos de software livre. Para isso, a ferramenta é responsável por coletar, filtrar e fazer *download* dos projetos, e executar o comando `git grep` com as expressões de busca dos bancos de dados que se deseja analisar. Após a execução deste processo, o programa apresenta uma aplicação web na qual é possível visualizar as bibliotecas encontradas nos projetos, as linhas do código em que as expressões de busca foram encontradas e indicar se de fato consiste em um resultado positivo, ou se trata-se de um falso-positivo.

Na sequência, serão detalhados cada um dos arquivos do código-fonte desta ferramenta. Os arquivos com extensão *.py* tratam-se de *scripts* desenvolvidos em linguagem Python, enquanto as extensões *.ipynb* consistem em Jupyter Notebooks[2] e os arquivos *.xlsx* são tabelas do programa Microsoft Office Excel.

O *script collect.py* é responsável por realizar a coleta dos metadados dos projetos, utilizando-se a API do GitHub com GraphQL (v4)[3], e salvá-los de forma legível no arquivo *projects.xlsx*. O *script filter.ipynb* é responsável por aplicar os filtros e registrar o novo corpo de projetos no arquivo *filtered.xlsx*. A ferramenta é projetada para que, nesta etapa, os projetos sejam analisados manualmente e identificados como sendo, ou não, um *software* voltado ao usuário final. Para isso, uma nova coluna, denominada *"Is A Software?"* é criada manualmente na tabela para indicar esta informação. Esta última versão da tabela contendo o corpo de projetos é dada pelo arquivo *annotated.xlsx*. Assim, este arquivo é utilizado pelo *script download.py*, responsável por fazer o *download* dos repositórios. Por fim, o arquivo *extract.py* executa os comandos `git grep` nos diretórios dos projetos e salva os resultados em um banco de dados relacional. O banco de dados a ser utilizado é indicado no arquivo *database.json*, podendo escolher entre SQLite e PostgreSQL. Por fim, o programa possui uma aplicação web, desenvolvida em React, a qual permite validar os resultados manualmente. A aplicação mostra as ocorrências encontradas entre bibliotecas e projetos, e para cada ocorrência, a saída gerada pelo do `git grep` é apresentada. É possível então, analisar os resultados e confirmar, ou não, as ocorrências.

Sendo assim, a ferramenta em questão foi adaptada para o contexto desta pesquisa, utilizando-se as expressões de busca correspondentes às bibliotecas de interface gráfica de usuário, e fazendo as adaptações necessárias para alcançar o objetivo proposto.

[1] https://github.com/gems-uff/db-mining
[2] https://jupyter.org/
[3] https://docs.github.com/en/free-pro-team@latest/graphql

4.4 Método de pesquisa

O método proposto nesta pesquisa foi dividido em 6 etapas. A primeira etapa consiste na definição das linguagens de programação que compõem o escopo da pesquisa. A segunda etapa contempla a seleção das bibliotecas de interface gráfica de usuário que foram examinadas. A terceira etapa consiste na definição do corpo de projetos de software livre que foram coletados, filtrados e analisados. A construção das expressões de busca das bibliotecas acontece na quarta etapa. Na quinta etapa, é executada a busca pelas bibliotecas no código-fonte dos projetos. A sexta etapa compreende o processo de mineração dos *commits* dos projetos. Na Figura 4.1, apresenta-se o fluxograma das etapas do método de pesquisa em questão. Cada uma das etapas são detalhadas nas subseções a seguir.

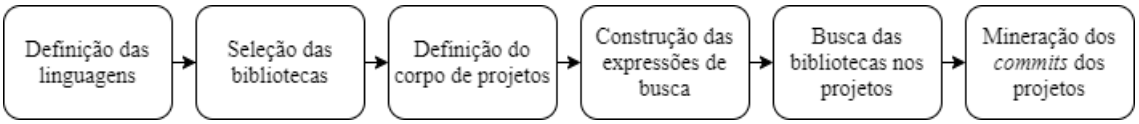

Figura 4.1. Etapas do Método de Pesquisa.

4.4.1 Definição das linguagens

Nesta etapa, foram definidas as linguagens de programação que irão fazer parte do escopo desta pesquisa. Para isso, selecionou-se seis linguagens de programação que se fazem presentes no índice TIOBE[4], um dos mais estimados indicadores de popularidade das linguagens de programação. Segundo TIOBE (2020), "o índice TIOBE é atualizado uma vez por mês. As classificações são baseadas no número de engenheiros qualificados em todo o mundo, cursos e fornecedores terceirizados. Motores de busca populares como Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube e Baidu são usados para calcular as classificações".

Sendo assim, das 20 linguagens de programação mais populares segundo o índice TIOBE, as linguagens selecionadas foram **C**, **C++**, **C#**, **Java**, **JavaScript** e **Python**. Neste caso, considerou-se o índice de Novembro de 2020, quando iniciou-se o desenvolvimento desta pesquisa. Vale ressaltar que as linguagens selecionadas não necessariamente correspondem às seis linguagens de programação mais bem colocadas. A escolha de tais linguagens foi feita baseando-se no índice TIOBE, e também no conhecimento prévio de que tais linguagens de programação possuem suporte à interfaces gráficas de usuário. Além disso, a escolha desta quantidade de linguagens justifica-se pelo fato de que, neste momento da pesquisa, inviabiliza-se a realização do estudo com todas as 20 linguagens presentes no índice. A Tabela 4.1 apresenta as linguagens de programação selecionadas e suas respectivas posições nos índices de popularidade TIOBE e Octoverse[5].

[4] https://www.tiobe.com/tiobe-index/
[5] https://octoverse.github.com/

Linguagem	Índice TIOBE	Índice Octoverse
C	1º	9º
Python	2º	2º
Java	3º	3º
C++	4º	6º
C#	5º	5º
JavaScript	7º	1º

Tabela 4.1. Linguagens de programação e suas respectivas posições nos índices de popularidade.

4.4.2 Seleção das bibliotecas

Nesta etapa, selecionou-se as principais bibliotecas de interface gráfica de usuário presentes nas linguagens de programação definidas na etapa anterior. Bem como no estudo preliminar, a escolha das bibliotecas foi realizada com base na literatura cinzenta.

Para efetuar a busca dos materiais, utilizou-se o mecanismo de pesquisa do Google. Para esta finalidade, as seguintes *strings* de busca foram construídas: *"The Best X GUI Libraries"*, *"Top GUI Libraries for X"* e *"The Most Popular X GUI Libraries"*, sendo que X representa o nome da linguagem de programação, e GUI corresponde à uma abreviatura para *Graphical User Interface* (em português, Interface Gráfica de Usuário). Os termos foram utilizados em inglês com objetivo de obter-se melhores resultados nas buscas.

Por exemplo, no que se refere à linguagem de programação Java, as *strings* de busca foram dadas como: *"The Best Java GUI Libraries"*, *"Top GUI Libraries for Java"* e *"The Most Popular Java GUI Libraries"*. Desta maneira, os cinco principais resultados retornados pelas buscas foram selecionados. Dentre eles, artigos de fóruns, blogs e sites de tecnologia e programação. Por fim, foram selecionadas as bibliotecas de interface gráfica de usuário que mais foram citadas nestes materiais. Este procedimento foi realizado para todas as linguagens do escopo desta pesquisa. Como resultado, foram selecionadas 42 bibliotecas ao total. A Tabela 4.2 apresenta a lista das 42 bibliotecas de interface gráfica de usuário selecionadas, separadas por linguagem de programação.

4.4.3 Definição do corpo de projetos

Após definidas as linguagens e selecionadas as bibliotecas, foi iniciado o processo de coleta, filtragem e seleção dos projetos de software livre. A seleção do corpo de projetos foi executada com foco em projetos de software livre bem estruturados e relevantes para a comunidade. Como já mencionado no objetivo da pesquisa, foram incluídos somente projetos de software livre hospedados na plataforma GitHub.

Inicialmente, foram buscados, no GitHub, todos os repositórios de projetos públicos que não fossem *forks* de outros repositórios, não estivessem arquivados, tendo no mínimo 1000 estrelas e pelo menos uma alteração (*push*) nos últimos três meses em relação à data da coleta. Para isso, utilizou-se o *script collect.py* da ferramenta de mineração mencionada anteriormente. A busca retornou um total de 7240 projetos, que foram exportados para o arquivo *projects.xlsx*.

C	C++	C#	Java	JavaScript	Python
Gtk	CEGUI	Avalonia	AWT	Angular	Flexx
IUP	FLTK	WindowsForms	JavaFX	AntDesign	Kivy
Win32	Gtk++	Xamarin	QtJambi	Blueprint	PyForms
	Nana		Swing	Buefy	PyGTK
	Qt		SWT	MaterialUI	PyGUI
	Sciter			ngx-bootstrap	PyQt
	Tk			OnsenUI	tKinter
	wxWidgets			PrimeNG	wxPython
				Quasar	
				React	
				ReactBootstrap	
				TeradataCovalent	
				Vue	
				VueMaterial	
				Vuetify	

Tabela 4.2. Lista das Bibliotecas Selecionadas e suas Respectivas Linguagens de Programação.

Segundo Kalliamvakou et al. (2014), evitar *forks* é importante para garantir que o corpo de projetos possua somente um repositório por projeto. Já a restrição de no mínimo 1000 estrelas, garante que sejam selecionados somente projetos que possuem alguma relevância para a comunidade (BORGES; VALENTE, 2018; LIMA; HORA, 2020). Além disso, evitar repositórios arquivados e repositórios que não possuem alguma alteração nos últimos três meses, garante um certo grau de atividade, de modo a descartar projetos descontinuados ou inativos.

Na sequência, foi utilizado o *script filter.ipynb*, a fim de aplicar filtros adicionais quanto ao número de contribuidores (no mínimo 5) e número de *commits* (no mínimo 1000 na *branch* principal). Tais filtros ajudaram a descartar projetos pessoais, projetos de pequeno-porte e projetos em fase inicial. Além disso, como definido na Seção 4.4.1, as linguagens de programação incluídas no escopo desta pesquisa foram C, C++, C#, Java, JavaScript e Python. Portanto, os repositórios também foram filtrados quanto à sua linguagem de programação principal. Projetos desenvolvidos em linguagens que não fossem uma das seis linguagens mencionadas, foram descartados. Após a aplicação dos filtros mencionados, o corpo de projetos foi reduzido de 7240 para 3746 projetos de software livre, que foram salvos no arquivo *filtered.xlsx*

Ao final, foi executada uma análise manual dos projetos e suas documentações, por meio da leitura do arquivo *README.md* e da página oficial do projeto (quando houvesse), com a finalidade selecionar apenas projetos que tivessem interface gráfica de usuário. Portanto, foram descartados projetos como *frameworks*, bibliotecas, linguagens de programação, compiladores e afins. Nesta etapa, definiu-se como objetivo, para compor o corpo de projetos final do estudo, selecionar um total de 30 projetos de cada uma das 6 linguagens de programação consideradas no escopo da pesquisa. Dessa forma, os projetos foram analisados por linguagem de programação e em ordem decrescente em relação ao número de estrelas. Isto é, começando pelos projetos mais bem estrelados. Quando identificados 30 projetos contendo interface gráfica de usuário dentro do contexto de uma determinada

linguagem de programação, iniciava-se então a busca dos projetos da próxima linguagem. Portanto, o corpo de projetos final possui um total de 180 projetos que foram salvos no arquivo *annotated.xlsx*. A Tabela 4.3 sumariza o processo de seleção dos projetos.

Linguagem	Após coleta	Após filtragem	Descartados (sem GUI)	Selecionados (com GUI)	Proporção
C	567	421	144	30	4.8
C++	701	603	76	30	2.5
C#	397	302	94	30	3.1
Java	1178	602	86	30	2.9
JavaScript	2809	947	192	30	6.4
Python	1588	871	110	30	3.7
Total	**7240**	**3746**	**702**	**180**	**3.9**

Tabela 4.3. Quantidade de projetos após cada etapa do processo de coleta, filtragem e análise manual.

4.4.4 Construção das expressões de busca

Para identificar se um projeto de software livre faz uso de uma determinada biblioteca, foi utilizado o comando `git grep`. Tal comando trata-se de um recurso provido pelo sistema de controle de versão distribuído, o `git`[6]. O comando `git grep` realiza uma busca por uma palavra ou padrão específico, dentro dos arquivos existentes em um diretório, e retorna as linhas onde o padrão foi encontrado. Nesta etapa do método de pesquisa, foram construídos os padrões de identificação das bibliotecas. Tais padrões foram denominados como *expressões de busca*.

As expressões de busca foram construídas considerando-se Raemaekers et al. (2012b), que investigaram a frequência do uso de bibliotecas de terceiros em um corpo de projetos de software livre e software proprietário, desenvolvidos em linguagem Java. Para isso, Raemaekers et al. (2012b) utilizaram como expressões de busca as sentenças de importação das bibliotecas, seguindo o padrão `import <nome_biblioteca>;`. Também foi considerado utilizar outros padrões, que não fossem somente a sentença de importação, mas que pudesse servir como uma expressão de busca complementar para identificar o uso da biblioteca.

Para construção das expressões de busca, as documentações das bibliotecas foram consultadas e foi buscado por projetos exemplo, de modo a extrair os padrões de importação e utilização de cada uma das bibliotecas. Após a identificação destes padrões, a sentença de importação ou palavra-chave foi convertida em uma expressão regular a ser utilizada como entrada para busca da biblioteca nos projetos. Por exemplo, a sentença de importação da biblioteca Gtk, da linguagem C foi identificada como sendo `#include <gtk/gtk.h>`, e a expressão complementar foi a palvra-chave `GtkApplication`. Assim, as expressões de busca finais dessa biblioteca foram, respectivamente: `[”<]gtk(/gtk)?(.h)?[”>]?` e `Gtk[a-zA-Z]*`. A partir da execução

[6] https://git-scm.com/

desta etapa, foi construída uma tabela contendo todas as expressões de busca a serem utilizadas na busca pelas bibliotecas.

A partir da tabela mencionada acima, foi desenvolvido um *script* para automatizar o processo de criação um arquivo texto com o nome da biblioteca, contendo na primeira linha a expressão de busca referente à sentença de importação da biblioteca, e na segunda linha a expressão de busca referente à heurística complementar, caso houvesse. Dessa forma, a partir da execução desse *script*, foi gerado um conjunto de 42 arquivos texto, sendo um para cada biblioteca a ser analisada. Cada arquivo foi nomeado com o nome da biblioteca, tendo em seu conteúdo suas respectivas expressões de busca. Por exemplo, se tratando da biblioteca AWT, da linguagem Java, foi gerado o arquivo *AWT.txt* cujo conteúdo é a expressão de busca `java.awt`. Esse processo foi necessário para adequar o formato de entrada das expressões busca ao padrão que já estava sendo utilizado pela ferramenta de mineração. Por sua vez, a ferramenta executa o comando `git grep` utilizando o parâmetro `-f` que recebe um arquivo texto e realiza a busca pelas expressões regulares existentes dentro do arquivo. Por exemplo, o comando completo a ser executado para buscar a biblioteca tKinter nos projetos Python foi `git grep -I –context=5 –break –heading –line-number –color=always –extended-regexp -f /resources/heuristics/libraries/python/tKinter.txt`.

4.4.5 Busca das bibliotecas nos projetos

Tendo realizado as etapas anteriores, executou-se então a busca pelas bibliotecas nos projetos de software livre. Para isso, inicialmente foi utilizado o *script download.py* para baixar o código-fonte dos 180 projetos a serem analisados. Após isso, foi adaptado o *script extract.py* para realizar a procura pelas expressões de busca das bibliotecas apenas nos projetos de suas respectivas linguagens. Sendo assim, uma mesma biblioteca foi buscada em cada um dos 30 projetos analisados em sua respectiva linguagem de programação.

Após a execução do *script extract.py*, obtém-se o resultado das buscas de cada uma das bibliotecas nos projetos de suas respectivas linguagens de programação. Tais resultados são registrados em um banco de dados relacional, no arquivo *database.sqlite*. Sendo assim, foi possível executar a interface web da ferramenta de mineração para validar os resultados manualmente, como no exemplo da Figura 4.2. A interface faz a leitura dos resultados gravados no banco de dados, e apresenta a listagem dos projetos e cada uma das bibliotecas encontradas no projeto em questão. Assim, cada um dos resultados apresentados na interface web foram validados manualmente, indicando se trata-se de um resultado positivo, ou se considera-se um falso-positivo. Dessa forma, todos os resultados produzidos nesta etapa foram exportados para o arquivo *first_round.csv*, para serem posteriormente analisados.

Tendo realizado a execução anterior, obteve-se os dados necessários para produzir as análises da popularidade das bibliotecas em termos da quantidade de ocorrências nos projetos. Sendo assim, as próximas etapas que serão detalhadas a seguir, correspondem à coleta dos dados necessários para

Figura 4.2. Exemplo de Saída do git grep na Interface Web para Validação Manual dos Resultados.

analisar-se a popularidade das bibliotecas em termos da quantidade de arquivos que as utilizam nos projetos.

Para contabilizar em quantos arquivos dos projetos de software livre as bibliotecas são importadas, optou-se por construir novas expressões de busca, dessa vez que coincidam exatamente com a sentença de importação da biblioteca. Por exemplo, na primeira rodada de execuções, a expressão de busca derivada da sentença de importação da biblioteca PyGTK, do Python, havia sido `("Gtk")|(Gtk)`. Nesta segunda rodada, entretanto, definiu-se como `(import|from) [Gg]tk[.\w]*`. Dessa forma, os arquivos texto contendo as expressões de busca das bibliotecas foram gerados novamente. Também foi alterado o *script* de busca da bibliotecas, adicionando-se o parâmetro `count` no comando `git grep`, com intuito de identificar a quantidade de arquivos do código-fonte do projeto que importam a biblioteca em questão. Portanto, o *script extract.py* foi executado novamente, e os resultados indicam em quantos arquivos uma biblioteca é importada dentro de um determinado projeto. Os resultados desta execução foram salvos no arquivo *database_2nd_round.sqlite* e foram extraídos para o arquivo *second_round.csv*.

Por fim, se fez necessário descobrir a quantidade total de arquivos de código-fonte dos projetos analisados. Para isso, foi utilizado o CLOC (*Count Lines of Code*)[7]. O programa em questão possui como principal recurso a contagem dos arquivos dentro de um diretório, descartando arquivos que não sejam código-fonte. Dessa forma, foi construído o *script count.py* para executar o CLOC dentro dos diretórios dos projetos e exportar os resultados para o arquivo *count.csv*.

Assim, todos os resultados obtidos com as execuções anteriores foram condensados em um único arquivo, denominado *results.xlsx*. Neste, a tabela final com os resultados possui, para cada linguagem de programação, o resultado da busca de cada biblioteca em cada um dos projetos de sua respectiva linguagem. Dessa forma, para cada busca, o campo *regexMatch* da tabela indica se a expressão de busca da biblioteca foi, ou não, encontrada naquele projeto. Nos casos em que a biblioteca

[7] https://github.com/AlDanial/cloc

foi identificada no projeto, há também o resultado da validação manual daquele caso, apontado pelo campo *manualValidation*, indicando se o resultado trata-se de um caso positivo, representado pelo valor 1, ou se trata-se de um falso-positivo, representado pelo valor 0. Além disso, tem-se a quantidade de arquivos nos quais a biblioteca foi identificada em determinado projeto, indicada pelo campo *countFilesRegex*, bem como a quantidade total de arquivos que o projeto possui, indicada no campo *clocTotalFiles*. Por exemplo, a linha da tabela que representa a busca pela biblioteca Qt, no projeto **tiled**, da linguagem C++, possui o valor 1 para o campo *regexMatch*, indicando que a expressão de busca dessa biblioteca foi encontrada neste projeto. No campo *manualValidation* também há o valor 1 indicando que o resultado foi validado como positivo durante a análise manual. Além disso, o campo *countFilesRegex* possui o valor 45, indicando a quantidade de arquivos nos quais a biblioteca Qt foi encontrada neste projeto. Por fim, o campo *clocTotalFiles* com o valor 1097 indica a quantidade total de arquivos de código-fonte que projeto **tiled** possui. A Figura 4.3 apresenta alguns exemplos das buscas da biblioteca Qt em projetos C++, com o exemplo mencionado anteriormente em destaque.

language	library	projectName	countFilesRegex	clocTotalFiles	regexMatch	manualValida
cpp	Qt	OpenRCT2	0	1207	0	0
cpp	Qt	polybar	0	282	0	0
cpp	Qt	PowerToys	0	2523	0	0
cpp	Qt	serenity	0	5620	0	0
cpp	Qt	tdesktop	0	1383	0	0
cpp	Qt	xbmc	0	6757	0	0
cpp	Qt	Sourcetrail	11	1735	1	1
cpp	Qt	keepassxc	13	995	1	1
cpp	Qt	sqlitebrowser	15	494	1	1
cpp	Qt	cutter	15	640	1	1
cpp	Qt	vnote	15	966	1	1
cpp	Qt	qBittorrent	15	1004	1	1
cpp	Qt	rpcs3	15	1142	1	1
cpp	Qt	robomongo	25	1268	1	1
cpp	Qt	FreeCAD	44	8561	1	1
cpp	Qt	tiled	45	1097	1	1
cpp	Qt	CRYENGINE	83	16239	1	1

Figura 4.3. Exemplos do Resultado Final de Buscas pela Biblioteca Qt em Projetos C++.

Dessa maneira, tem-se os dados necessários para responder a primeira questão de pesquisa do presente estudo, que diz respeito à popularidade das bibliotecas de interface gráfica de usuário nos projetos de software livre.

4.4.6 Mineração dos *commits* dos projetos

Nesta subseção são apresentados os detalhes do processo de mineração dos *commits* dos projetos. Este processo corresponde ao levantamento dos dados necessários para realizar-se as análises referentes à segunda questão de pesquisa deste estudo, com a qual busca-se explorar as relações das bibliotecas de interface gráfica de usuário com as demais partes do código-fonte dos projetos de software livre.

Na primeira questão de pesquisa do presente estudo, concluiu-se que as bibliotecas de interface gráfica de usuário mais populares de suas respectivas linguagens de programação foram Win32 (C), WindowsForms (C#), Qt (C++), AWT (Java), React (JavaScript) e PyQt (Python). Sendo assim, utilizou-se tais bibliotecas como base para a execução das análises referentes à segunda questão de pesquisa. Para isso, foram selecionados os projetos de software livre nos quais as bibliotecas em questão foram encontradas. Por exemplo, no caso da linguagem Java, selecionou-se todos os projetos nos quais a biblioteca AWT foi encontrada. Fazendo isso para as 6 linguagens de programação, foi obtido um total de 93 projetos distribuídos entre elas. A partir de então, foi construído um *script* para minerar os *commits* dos projetos e extrair aqueles que possuem alterações nos arquivos em que a biblioteca foi encontrada.

O *script* de coleta dos *commits* foi desenvolvido em linguagem Python, utilizando-se o PyDriller, que é um *framework* cujo propósito é auxiliar desenvolvedores no processo de extração de informações de repositórios Git (SPADINI et al., 2018). O PyDriller provê recursos para obter-se informações sobre *commits*, desenvolvedores, arquivos modificados e código-fonte dos projetos de maneira simples e fácil. Dessa forma, o *script* de busca dos *commits* foi responsável executar os seguintes passos:

1. Obter a listagem dos projetos de software livre nos quais as bibliotecas mais populares foram encontradas.
2. Para cada projeto:

    2.1. Obter a lista dos arquivos em que a biblioteca é importada, a partir dos dados gerados na etapa de busca das expressões regulares nos projetos.
    2.2. Extrair todos os *commits* do projeto.
    2.3. Verificar se há alteração de algum dos arquivos obtidos no Passo 2.1 em cada um dos *commits* do projeto.
    2.4. Caso haja alteração de algum dos arquivos no *commit* analisado, salvar as informações do mesmo em um arquivo *csv*. Tais informações são código *hash*, autor, mensagem, data, linhas adicionadas, alteradas e removidas, e listagem dos arquivos modificados no *commit*.

Dessa forma, após a execução do *script* de busca dos *commits*, obteve-se um arquivo *csv* para cada projeto analisado, em que o conteúdo do arquivo contém a lista dos *commits* que possuem alguma alteração de arquivos que importam a biblioteca mais popular daquela linguagem. Por exemplo, para o projeto **notepad-plus-plus**, da linguagem C++, obteve-se o arquivo *notepad-plus-plus_notepad-plus-plus.csv*, o qual contém as informações de 5 *commits* diferentes. Neste caso, significa que os *commits* em questão possuem alguma alteração em arquivos que importam a biblioteca Qt, a mais popular dentre os projetos C++.

Na sequência, definiu-se como objetivo analisar manualmente 50 *commits* dentre os projetos de cada linguagem de programação. Dessa forma, os *commits* foram selecionados aleatoriamente, distribuindo-os proporcionalmente em relação ao total de *commits* obtidos por projeto. Isto é, projetos com maiores números de *commits* também tiveram mais *commits* analisados, enquanto analisou-se

menos *commits* dos projetos com menores números. Por exemplo, analisou-se 16 dos 1137 *commits* do projeto Arduino, da linguagem Java, enquanto foram analisados 3 dos 197 *commits* do projeto zaproxy da mesma linguagem. Assim, para cada linguagem de programação, foi gerada uma lista com os 50 *commits* a serem analisados, contendo suas respectivas informações e projeto ao qual pertence.

A partir de então, executou-se a análise manual dos *commits* com objetivo de explorar padrões de utilização das bibliotecas nos projetos. Para isso, definiu-se os seguintes critérios para serem avaliados durante o processo de análise manual: número de arquivos modificados, tipo do *commit*, se possui referência à interface de usuário ou à biblioteca na mensagem e se possui alteração de outros arquivos que não sejam de interface de usuário. Em relação aos tipos de *commits*, foram definidos 6 tipos principais com base em Goyal et al. (2018). São eles: *fix*, *change*, *feature*, *refactor*, *release* e *build*. A Tabela 4.4 apresenta as descrições de cada um dos tipos de *commits* mencionados.

Tipo	Descrição
Fix	Alteração cujo objetivo seja corrigir algum problema ou bug no projeto.
Change	Alteração de propósito geral que não se encaixe em alguma categoria específica.
Feature	Alteração cujo objetivo seja implementar um novo recurso do projeto.
Refactor	Alteração para refatorar alguma parte do código-fonte do projeto.
Release	Alteração cujo objetivo seja lançar uma nova versão do projeto.
Build	Alteração para configurar os arquivos referentes à construção do projeto.

Tabela 4.4. Descrições dos tipos de *commits*.

Já em relação aos *commits* que possuem alterações em arquivos que não sejam de interface de usuário, buscou-se explorar e anotar à quais partes do projeto pertencem esses arquivos. Neste caso, as alterações foram classificadas em 4 diferentes tipos, sendo eles *feature*, *backend*, *tests* e *many*. A Tabela 4.5 apresenta a descrição de cada uma das classificações mencionadas.

Tipo	Descrição
Feature	Alteração em arquivos relacionados à algum recurso ou funcionalidade do projeto.
Backend	Alteração em arquivos que fazem parte do backend do projeto.
Tests	Alteração em arquivos de testes.
Many	Alterações em muitos arquivos diferentes.

Tabela 4.5. Descrições dos tipos de alterações em arquivos que não sejam da interface gráfica de usuário do projeto.

Desta maneira, a análise manual dos *commits* foi executada da seguinte forma: dentro do escopo de cada linguagem de programação, utilizou-se a lista contendo as informações dos 50 *commits* a serem analisados. Para cada um deles, acessou-se a página do GitHub que apresenta as informações do *commit* e, a partir disso, as observações foram anotadas em uma planilha seguindo os critérios mencionados anteriormente (número de arquivos modificados, tipo do *commit*, se possui referência à interface de usuário ou à biblioteca na mensagem e se possui alteração de outros arquivos que não sejam de interface de usuário). Assim, obteve-se os dados necessários para responder a QP2 do

presente estudo. Os detalhes e análises referentes à esta questão de pesquisa são apresentados na Seção 5.2.

Todos os arquivos referentes à execução do método de pesquisa estão disponíveis no repositório `lucasvribeiro/tcc`, no GitHub[8].

[8] https://github.com/lucasvribeiro/tcc

5 RESULTADOS

Este capítulo apresenta os resultados obtidos que respondem as questões de pesquisa e objetivo definidos no Capítulo 4. Em seguida, as implicações deste estudo são explicitadas e, por fim, apresenta-se as ameaças à validade.

5.1 QP1: Quais são as bibliotecas de interface gráfica de usuário mais populares entre as 6 linguagens de programação analisadas?

Conforme apresentado no Capítulo 4, no presente estudo foram consideradas 42 bibliotecas de 6 linguagens de programação diferentes. Inicialmente, observou-se que 16 bibliotecas não tiveram suas expressões de busca encontradas em qualquer projeto de sua respectiva linguagem. Dessa forma, as bibliotecas em questão foram descartadas das próximas análises, uma vez que não possuíram qualquer ocorrência nos projetos. Pode-se concluir, portanto, que tais bibliotecas possuem baixa popularidade dentre os projetos de software livre de suas respectivas linguagens de programação. A Tabela 5.1 apresenta a lista com as 16 bibliotecas descartadas.

Linguagem	Biblioteca
C++	CEGUI
C++	Nana
C++	Sciter
C++	Tk
Java	QtJambi
JavaScript	AntDesign
JavaScript	Blueprint
JavaScript	Buefy
JavaScript	ngx-bootstrap
JavaScript	OnsenUI
JavaScript	PrimeNG
JavaScript	VueMaterial
JavaScript	Vuetify
Python	Kivy
Python	PyGTK
Python	wxPython

Tabela 5.1. Listagem das 16 Bibliotecas Descartadas por Não Possuírem Qualquer Ocorrência nos Projetos.

Após a busca pelas expressões de busca das bibliotecas nos projetos de suas respectivas linguagens, os resultados obtidos foram validados manualmente por meio da interface web na ferramenta de mineração. Sendo assim, após descartar as bibliotecas mencionadas anteriormente,

foram descartadas as bibliotecas que tiveram alguma ocorrência nos projetos, mas todas foram descartadas no processo de validação manual. Isto é, bibliotecas que tiveram alguma ocorrência durante a busca das expressões de busca nos projetos, mas todas as ocorrências foram caracterizadas como falsos-positivos na etapa de validação manual. Nesse caso, observou-se 8 bibliotecas com essas características, as quais também foram descartadas das próximas análises. Na Tabela 5.2, apresenta-se a lista das 8 bibliotecas em questão.

Linguagem	Biblioteca
C	IUP
C++	FLTK
Java	JavaFX
JavaScript	Quasar
JavaScript	TeradataCovalent
Python	Flexx
Python	PyForms
Python	PyGUI

Tabela 5.2. Listagem das 8 Bibliotecas Descartadas Após a Validação Manual.

Vale ressaltar que, durante a fase de validação manual dos resultados, observou-se grande quantidade de falsos-positivos, isto é, casos em que a expressão de busca da biblioteca foi encontrada em um determinado projeto, no entanto, essa ocorrência foi descartada na etapa de análise manual.

Dentre os projetos de linguagem C, a biblioteca com maior número de falsos-positivos foi a Gtk, tendo 9 (30%) casos contabilizados. Em relação aos projetos C++, a biblioteca wxWidgets apresentou o maior número de falsos-positivos, com 9 (30%) ocorrências. Já dentre os projetos de linguagem C#, observou-se 17 (56,6%) casos de falsos-positivos da biblioteca Xamarin. Dos projetos Java, a biblioteca Swing apresentou o maior número de falsos-positivos, tendo 3 (10%) ocorrências observadas. Dentre os projetos JavaScript, a biblioteca Angular apresentou 20 (66,6%) casos de falsos-positivos. Por fim, em relação aos projetos de linguagem Python, a biblioteca com maior número de casos de falsos-positivos foi a PyQt, com 18 (60%) ocorrências. A Tabela 5.3 apresenta a lista de todos os falsos-positivos, agrupados por biblioteca e linguagem de programação.

Alguns exemplos de falsos-positivos se deram com as expressões de busca sendo encontradas em *URLs*, funções *hash*, comentários de código e outros blocos de texto que não compreendem código-fonte. As Figuras 5.1, 5.2 e 5.3 apresentam alguns exemplos destes casos. Na Figura 5.1, a expressão de busca da biblioteca PyQt foi encontrada no campo *integrity* de uma dependência no arquivo *package-lock.json*. No exemplo da Figura 5.2, apresenta-se um falso-positivo da biblioteca Angular, do JavaScript, encontrada em um arquivo *JSON* que representa um emoji. A Figura 5.3 apresenta a expressão de busca da biblioteca Xamarin sendo encontrada em um arquivo *gitignore* de um projeto em linguagem C#.

Acredita-se que os números expressivos de falsos-positivos se deram por conta das expressões de busca terem sido construídas da forma mais abrangente possível. O objetivo foi reduzir ao máximo

Linguagem	Biblioteca	# de Falsos Positivos	%
C	**Gtk**	**9**	**30,0**
C	Win32	4	13,3
C	IUP	1	3,3
C++	**wxWidgets**	**9**	**30,0**
C++	FLTK	3	10,0
C#	**Xamarin**	**17**	**56,6**
C#	WindowsForms	4	13,3
C#	Avalonia	1	3,3
Java	**Swing**	**3**	**10,0**
Java	AWT	1	3,3
Java	JavaFX	1	3,3
JavaScript	**Angular**	**20**	**66,6**
JavaScript	Vue	6	20,0
JavaScript	React	3	10,0
JavaScript	Quasar	3	10,0
JavaScript	TeradataCovalent	1	3,3
Python	**PyQt**	**18**	**60,0**
Python	PyGUI	6	20,0
Python	tKinter	4	13,3
Python	Flexx	1	3,3
Python	PyForms	1	3,3

Tabela 5.3. Listagem dos Falsos-positivos por Biblioteca e Linguagem de Programação.

Figura 5.1. Exemplo de Falso-positivo em Arquivo *package-lock.json*.

os falsos-negativos, que seriam casos em que há uso de determinada biblioteca em um projeto, mas não tenha sido identificado pelas expressões de busca.

Assim sendo, das 42 bibliotecas consideradas inicialmente, 24 foram descartadas nas etapas mencionadas nos parágrafos anteriores, o que compreende a aproximadamente 57% do total. Dessa forma, é garantido que as 18 bibliotecas restantes tiveram alguma ocorrência positiva dentre os projetos de suas respectivas linguagens de programação. Na Tabela 5.4 apresenta-se a lista das 18 bibliotecas, suas respectivas linguagens de programação e a quantidade de projetos nos quais foram encontradas.

Figura 5.2. Exemplo de Falso-positivo em Arquivo *JSON* que representa um emoji.

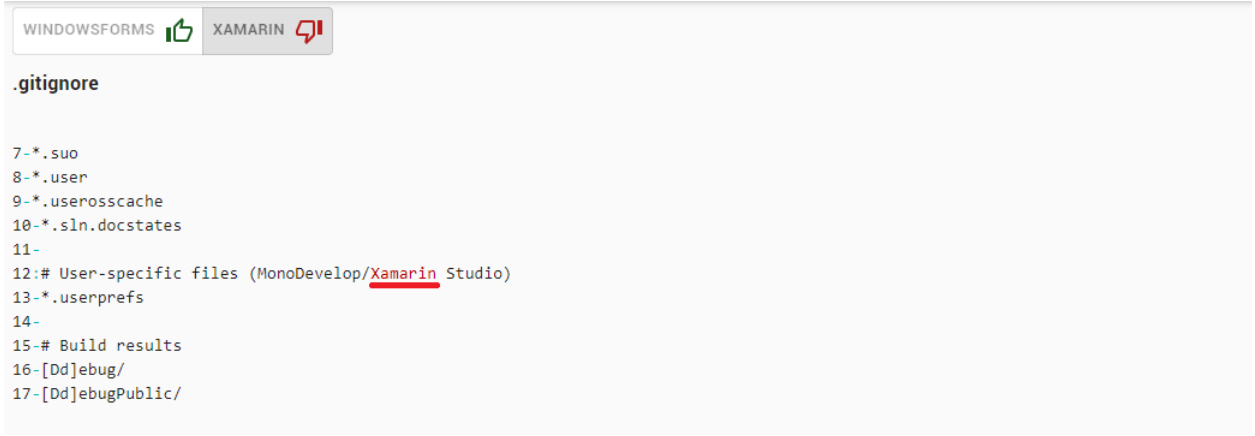

Figura 5.3. Exemplo de Falso-positivo em Arquivo *gitignore*.

Diante dos resultados apresentados, observou-se que a biblioteca com maior ocorrência entre os projetos de sua linguagem foi a Win32, sendo encontrada em 25 dos 30 projetos de linguagem C, o que corresponde a um total de 83,3%. Neste caso, é importante salientar que a biblioteca possui módulos e componentes não somente de interface gráfica de usuário, o que pode representar uma ameaça aos resultados referentes à esta biblioteca. Observou-se também que a Qt foi a biblioteca mais popular da linguagem C++, sendo encontrada em 53,3% dos projetos. Em relação aos projetos C#, foi possível observar a popularidade da biblioteca WindowsForms, encontrada em 19 dos 30 projetos, o que corresponde a um total de 63,3%. Se tratando dos projetos de linguagem Java, a biblioteca AWT foi a mais popular, sendo encontrada em 16 dos 30 projetos. Os resultados mostram ainda que a biblioteca React foi a mais popular entre os projetos JavaScript, encontrada em 46,7% dos projetos. Por fim, a biblioteca PyQt teve 13,3% de ocorrência nos projetos e foi a mais popular da linguagem Python.

Observou-se também que, das 18 bibliotecas, 5 delas tiveram ocorrência em mais de 50% dos projetos de sua linguagem, sendo elas a Win32 (C), Qt (C++), WindowsForms (C#) e AWT (Java). Além disso, observa-se que a biblioteca PyQt, a mais popular dentre os projetos Python, foi encontrada em

Linguagem	Biblioteca	# de Projetos	%
C	**Win32**	**25**	**83,3**
C	Gtk	12	40,0
C++	**Qt**	**16**	**53,3**
C++	Gtk++	2	6,7
C++	wxWidgets	2	6,7
C#	**WindowsForms**	**19**	**63,3**
C#	Xamarin	5	16,7
C#	Avalonia	1	3,3
Java	**AWT**	**16**	**53,3**
Java	Swing	11	36,7
Java	SWT	2	6,7
JavaScript	**React**	**13**	**43,3**
JavaScript	Angular	2	6,7
JavaScript	MaterialUI	1	3,3
JavaScript	ReactBootstrap	1	3,3
JavaScript	Vue	1	3,3
Python	**PyQt**	**4**	**13,3**
Python	tKinter	1	3,3

Tabela 5.4. Lista das 18 Bibliotecas Consideradas na Análise Final dos Resultados.

apenas 13,3% dos projetos. A hipótese é de que boa parte dos projetos analisados tenham interfaces gráficas web e, portanto, não fazem uso de bibliotecas da linguagem Python.

Como já mencionado na metodologia, o uso das bibliotecas também foi observado em termos da quantidade de arquivos as utilizam nos projetos analisados. Nesse sentido, buscou-se explorar este contexto com objetivo de analisar a popularidade das bibliotecas sob uma outra perspectiva.

Em relação à quantidade de arquivos que contêm a importação de uma determinada biblioteca, foi possível observar que, dentre os projetos de linguagem C, a maior taxa de importação de uma biblioteca foi no projeto **rufus**, com 11,14% dos arquivos importando a biblioteca Win32. Além disso, das 5 maiores ocorrências, 3 delas compreendem o uso da biblioteca Win32 e 2 ocorrências da biblioteca Gtk, o que confirma a popularidade de ambas as bibliotecas dentre os projetos de linguagem C. Já se tratando dos projetos de linguagem C++, as 5 maiores ocorrências de uma biblioteca em termos de arquivos do projeto compreendem o uso da biblioteca Qt. A maior ocorrência da biblioteca em questão foi no projeto **tiled**, com um total de 45 arquivos, o que corresponde a 4,10% do total de arquivos do projeto.

Em relação aos projetos de linguagem C#, as quatro principais ocorrências correspondem ao uso da biblioteca WindowsForms. A principal delas é o projeto **gitextensions** com 20,48% dos arquivos do projeto utilizando a biblioteca. A quinta posição consiste no projeto **MvvmCross**, utilizando a biblioteca Xamarin em 109 dos 2.493 arquivos do projeto. A Tabela 5.5 apresenta as 5 maiores ocorrências de bibliotecas em relação à proporção de arquivos do projeto, agrupados por linguagem de programação.

Linguagem	Biblioteca	Projeto	Arquivos com a Biblioteca	Total de Arquivos	Procentagem
C	**Win32**	**rufus**	**46**	**413**	**11,14**
C	Gtk	darktable	105	1071	9,80
C	Win32	reactos	718	20636	3,48
C	Win32	exploitdb	567	16907	3,35
C	Gtk	transmission	36	1107	3,25
C++	**Qt**	**tiled**	**45**	**1097**	**4,10**
C++	Qt	zeal	4	115	3,48
C++	Qt	sqlitebrowser	15	494	3,04
C++	Qt	cutter	15	640	2,34
C++	Qt	robomongo	25	1268	1,97
C#	**WindowsForms**	**gitextensions**	**357**	**1743**	**20,48**
C#	WindowsForms	mRemoteNG	126	752	16,76
C#	WindowsForms	subtitleedit	213	1340	15,90
C#	WindowsForms	ShareX	278	2174	12,79
C#	Xamarin	MvvmCross	109	2493	4,37
Java	**Swing**	**zaproxy**	**396**	**1578**	**25,10**
Java	Swing	material-theme-jetbrains	115	483	23,81
Java	AWT	material-theme-jetbrains	110	483	22,77
Java	AWT	zaproxy	334	1578	21,17
Java	SWT	dbeaver	763	4710	16,20
JavaScript	**React**	**reactotron**	**118**	**244**	**48,36**
JavaScript	React	carbon	71	150	47,33
JavaScript	React	wp-calypso	3516	11601	30,31
JavaScript	React	keystone-classic	249	885	28,14
JavaScript	React	Rocket.Chat	1167	4552	25,64
Python	**tKinter**	**faceswap**	**26**	**249**	**10,44**
Python	PyQt	DeepFaceLab	9	134	6,72
Python	PyQt	anki	8	1351	0,59
Python	PyQt	Mailpile	1	647	0,15
Python	PyQt	calibre	3	2604	0,12

Tabela 5.5. Top 5 Ocorrências de Maior Proporção nos Arquivos do Projeto por Linguagem de Programação.

As duas maiores ocorrências de bibliotecas de linguagem Java compreendem projetos utilizando a biblioteca Swing. O primeiro caso é o projeto **zaproxy**, tendo 396 arquivos utilizando a biblioteca, o que corresponde a 25,1% do total de arquivos do projeto. Por sua vez, a terceira e quarta ocorrência são projetos utilizando a biblioteca AWT. Já a quinta ocorrência foi da biblioteca SWT, a qual teve ocorrência em apenas 2 dos 30 dos projetos Java.

Dentre as bibliotecas de linguagem JavaScript, os projetos com maiores proporções de arquivos de código-fonte com importação da biblioteca foram projetos utilizando a biblioteca React. O maior caso foi o projeto **reactotron**, com 48,36% dos arquivos importando a biblioteca em questão. Por fim, em relação à linguagem de programação Python, o projeto com maior ocorrência de alguma biblioteca foi o **faceswap**, com 10,44% dos arquivos importando a biblioteca tKinter. Os outros 4 projetos subsequentes apresentam ocorrências da biblioteca PyQt.

Dessa forma, com base nos resultados obtidos, conclui-se que Win32 (C), WindowsForms (C#), Qt (C++), AWT (Java), React (JavaScript) e PyQt (Python) são as bibliotecas de interface gráfica de usuário mais populares por linguagem de programação, em termos de quantidade de projetos nos quais foram encontradas. Destaca-se ainda as bibliotecas Swing (Java) e tKinter (Python) que tiveram

resultados consideráveis em termos de quantidade de arquivos com importações das bibliotecas nos projetos.

Dentre as razões que influenciam na adoção de bibliotecas, Larios-Vargas et al. (2020) apontam que maturidade, estabilidade, boa documentação, usabilidade e comunidade ativa são fatores considerados por desenvolvedores no momento da escolha de uma biblioteca. Diante disso, justifica-se o fato das bibliotecas Win32 (C), Qt (C++), WindowsForms (C#), AWT (Java), React (JavaScript) e PyQt (Python) terem sido as mais populares de suas respectivas linguagens de programação. A biblioteca React, do JavaScript, por exemplo, possui boa documentação e exemplos, existe há um período considerável de tempo (aproximadamente 7 anos) e é mantida por uma grande comunidade e também por uma grande empresa, o Facebook. Já a biblioteca AWT, do Java, foi lançada pela primeira vez em 1995, juntamente com a própria linguagem, o que vai de encontro aos fatores de maturidade e estabilidade considerados pelos desenvolvedores. A biblioteca Qt, considerada neste estudo como a mais popular entre os projetos C++ e também nos projetos Python (PyQt), é uma biblioteca existente desde 1995 e mantida pela QtCompany.

Os resultados apresentados nesta questão de pesquisa produzem indicativos importantes em relação à popularidade das bibliotecas de interface gráfica de usuário dentro do universo de sua respectiva linguagem de programação. A popularidade é um dos fatores citados por Larios-Vargas et al. (2020) como sendo relevantes aos desenvolvedores no momento de escola de uma biblioteca. É importante salientar que outros fatores como tipo do projeto, momento da escolha da biblioteca e percepções individuais também podem influenciar na adoção das bibliotecas em projetos de software (LARIOS-VARGAS et al., 2020).

Quais são as bibliotecas de interface gráfica de usuário mais populares entre as 6 linguagens de programação analisadas?
As bibliotecas mais populares foram Win32 (C), Qt (C++), WindowsForms (C#), AWT (Java), React (JavaScript) e PyQt (Python).

5.2 QP2: Como as bibliotecas de interface gráfica de usuário se relacionam com as demais partes do código-fonte dos projetos?

A partir das análises realizadas na primeira questão de pesquisa, utilizou-se as bibliotecas mais populares para executar o estudo referente à **QP2**. Portanto, foram minerados os *commits* dos projetos que utilizam a biblioteca mais popular de sua respectiva linguagem. Dessa forma, para cada projeto, extraiu-se a lista dos *commits* que possuem alguma alteração em pelo menos um arquivo que utiliza a biblioteca. Então, selecionou-se 50 *commits* por linguagem de programação, distribuídos aleatoriamente entre os projetos. Dessa maneira, analisou-se manualmente os *commits*, a fim de

identificar a quantidade de arquivos alterados, se a mensagem de *commit* possui referência explícita à interface de usuário ou à biblioteca, se há alteração em outros arquivos que não tenham utilização da biblioteca e em quais tipos de arquivos isso aconteceu. Os resultados observados são apresentados nas subseções a seguir.

5.2.1 Projetos de Linguagem C

Em relação à linguagem de programação C, foram minerados os 25 projetos que utilizam a biblioteca Win32. Neste contexto, observou-se que a média de arquivos alterados por *commit* foi de 8 arquivos (mediana = 3, desvio padrão = 16). Diante disso, é possível observar que, de modo geral, os *commits* que alteram arquivos que utilizam a biblioteca Win32 geralmente possuem modificações em poucos arquivos do projeto.

Já em relação às características dos *commits* analisados, observou-se que 38 *commits* envolvendo o uso da biblioteca Win32, correspondem à mudanças gerais no código-fonte do projeto (*change*). Enquanto isso, 4 foram *commits* do tipo *fix*, que compreendem a correção de algum problema, 5 *commits* contendo a implementação de novos recursos (*feature*) e 2 *commits* referentes à *releases* de projeto. Somente 1 dos 50 *commits* observados possuem menção explícita à interface de usuário em sua mensagem. A Tabela 5.6 sumariza os resultados das análises desta linguagem.

Vale ressaltar que, os *commits* caracterizados como mudanças gerais no código-fonte do projeto, correspondem à alterações diversas que não puderam ser caracterizadas como qualquer outro tipo de alteração, como *fix*, *feature*, *refactor*, *release* ou *build*.

Como já mencionado, a biblioteca Win32 possui não somente elementos de interface gráfica de usuário. Diante da limitação de tempo para execução deste estudo, não foi possível observar se os *commits* analisados se tratam de alterações envolvendo elementos de interface gráfica de usuário ou se compreendem alterações utilizando outros módulos da biblioteca.

Tipos de commits	
Fix	4
Change	38
Feature	5
Refactor	0
Release	3
Build	0

Arquivos alterados	
Mínimo	1
Máximo	99
Mediana	3
Média	8

Mensagem com menção à UI	1

Commits com alteração de arquivos não-UI**	
Feature	2
Many	9
Backend	0
Tests	3

Commits com alteração de arquivos não-UI**	14

Tabela 5.6. Resumo dos *commits* dos projetos C.

** Não foi possível observar se os *commits* envolvendo o uso da biblioteca Win32 compreendem alterações na interface de usuário ou em outras partes do projeto.

5.2.2 Projetos de Linguagem C++

A respeito da linguagem C++, os 50 *commits* analisados foram distribuídos entre os 16 projetos nos quais a biblioteca Qt foi encontrada. Neste contexto, foi possível observar que a média de arquivos alterados por *commit* envolvendo o uso da biblioteca foi de 23 arquivos (mediana = 5, desvio padrão = 78). Observa-se, neste caso, que os dados em termos de quantidade de arquivos alterados nos *commits* são ligeiramente heterogêneos e pouco conclusivos para encontrar algum padrão de alteração envolvendo a biblioteca Qt.

Dentre os 50 *commits*, 26 foram caracterizados como mudanças gerais no código-fonte (*change*), 13 foram correção de algum problema (*fix*), 10 foram criação de novos recursos (*feature*) e 1 *commit* foi refatoração de código. Além disso, observou-se que 18 *commits* possuíram alterações em outras partes do projeto que não fossem interface de usuário. Nesse caso, 12 foram *commits* de alterações em algum recurso do sistema (*feature*), 3 alterações em muitos arquivos diferentes (*many*), 2 alterações em *back-end* e 1 *commit* de alteração em arquivos de testes. Além disso, observou-se que 8 *commits* possuem menção à interface de usuário em sua mensagem, o que facilita a identificação de *commits* referentes à alterações na interface de usuário do projeto. A Tabela 5.7 apresenta a sumarização dos dados observados após analisar os *commits* dos projetos C++.

Os dados observados referentes ao uso da biblioteca Qt nos projetos C++ sugerem que há uma baixa relação entre a biblioteca mencionada com arquivos de testes ou do *back-end* do sistema. Por outro lado, 24% dos *commits* que foram analisados possuíram alterações em outros recursos do projeto (*feature*), o que aponta para uma possível relação entre arquivos de interface de usuário que utilizam a biblioteca Qt, com o recurso do projeto que está sendo modificado no *commit*. Isso pode ser um indicativo de que os contribuidores dos projetos, em determinados momentos, terão que trabalhar em recursos do sistema de forma conjunta à modificações em sua interface gráfica de usuário. Isto é, uma tarefa relacionada à alguma alteração na interface de usuário do projeto pode demandar conhecimento dos recursos e funcionalidades que o projeto possui. Ainda assim, observa-se que a maioria das alterações analisadas compreendem mudanças somente na interface gráfica de usuário do projeto (64%).

Tipos de commits	
Fix	13
Change	26
Feature	10
Refactor	1
Release	0
Build	0

Arquivos alterados	
Mínimo	1
Máximo	515
Mediana	5
Média	23
Mensagem com menção à UI	8

Commits com alteração de arquivos não-UI	
Feature	12
Many	3
Backend	2
Tests	1
Commits com alteração de arquivos não-UI	18

Tabela 5.7. Resumo dos *commits* dos projetos C++.

5.2.3 Projetos de Linguagem C#

Sobre os resultados observados da linguagem C#, foram minerados os *commits* dos 19 projetos que utilizam a biblioteca WindowsForms. Dos 50 *commits* analisados, a média de arquivos alterados por *commit* foi de 6 arquivos (mediana = 3, desvio padrão = 17). Neste caso, observa-se que os *commits* que possuem modificações em arquivos que utilizam a biblioteca WindowsForms geralmente correspondem à alterações em poucos arquivos do projeto.

Se tratando das características dos *commits* analisados, observa-se que 26 foram caracterizados como *change*, enquanto 13 foram correções (*fix*), 10 *commits* com implementação de novos recursos (*feature*) e 1 *commit* de refatoração de código. Além disso, foi possível observar que 12 *commits* tiveram alterações de arquivos de outras partes do projeto, sendo 7 *commits* com alterações em alguma funcionalidade (*feature*), 4 alterações em muitos arquivos distintos (*many*) e 3 *commits* com alterações em arquivos de testes. A Tabela 5.8 sumariza os dados obtidos após analisar os *commits* desta linguagem.

Diante dos dados relacionados ao uso da biblioteca WindowsForms nos projetos C#, os indicativos são de que a biblioteca possui baixa relação com arquivos de testes, *back-end* e funcionalidades do sistema. Apenas 24% dos *commits* analisados tiveram alterações em outras partes do projeto que não fossem relacionadas à interface gráfica de usuário. Mais de três quartos (76%) dos *commits* tiveram modificações apenas em arquivos envolvendo a interface de usuário do projeto.

Tipos de commits	
Fix	11
Change	28
Feature	4
Refactor	7
Release	0
Build	0

Arquivos alterados	
Mínimo	1
Máximo	119
Mediana	3
Média	6
Mensagem com menção à UI	2

Commits com alteração de arquivos não-UI	
Feature	7
Many	4
Backend	0
Tests	3
Commits com alteração de arquivos não-UI	12

Tabela 5.8. Resumo dos *commits* dos projetos C#.

5.2.4 Projetos de Linguagem Java

Em relação às análises de linguagem Java, minerou-se os *commits* dos 16 projetos que utilizam a biblioteca AWT. A média de arquivos alterados por *commit* foi de 71 arquivos (mediana = 4, desvio padrão = 383). O máximo foi de 2705 arquivos alterados em um mesmo *commit*, o qual tratou-se de uma alteração de *release* do projeto. Neste caso, os dados se apresentam heterogêneos em termos de quantidade de arquivos alterados por *commit*, sendo pouco conclusivos neste contexto.

Dos 50 *commits* analisados, 21 deles tiveram alterações de outras partes do projeto que não fossem interface de usuário. Destes, 7 *commits* possuíram alterações em arquivos relacionados a algum recurso do projeto (*feature*), enquanto 9 *commits* tiveram alterações em muitos arquivos

distintos (*many*). Além disso, 5 *commits* tiveram modificações em arquivos de testes e 2 *commits* alteraram arquivos de *back-end* do projeto. Os dados obtidos a partir das análises dos *commits* de projetos Java são apresentados na Tabela 5.9.

Diante do exposto, observa-se que 10% dos *commits* envolveram alterações em arquivos de testes. Apesar de pequeno, é um resultado de alguma expressão e que pode ser melhor explorado em outras pesquisas, no sentido de analisar, dentro desse contexto, a relação entre alterações envolvendo a biblioteca AWT e arquivos de testes. Além disso, observa-se a baixa relação de arquivos de interface de usuário, que utilizam a biblioteca AWT com o *back-end* do projeto. De modo geral, observa-se que quase metade (42%) dos *commits* analisados também possuem alterações em arquivos de outras partes do projeto. Este pode ser um indicativo de que tarefas envolvendo modificações na interface de usuário também demandam alterações em outras partes do projeto.

Tipos de commits	
Fix	18
Change	23
Feature	1
Refactor	7
Release	1
Build	0

Arquivos alterados	
Mínimo	1
Máximo	2.705
Mediana	4
Média	71

Mensagem com menção à UI	5

Commits com alteração de arquivos não-UI	
Feature	7
Many	9
Backend	2
Tests	5

Commits com alteração de arquivos não-UI	21

Tabela 5.9. Resumo dos *commits* dos projetos Java.

5.2.5 Projetos de Linguagem JavaScript

Já em relação aos resultados obtidos com a linguagem JavaScript, os *commits* foram selecionados dentre os 13 projetos que utilizam a biblioteca React. Dos 50 *commits* analisados, a média de arquivos alterados por *commit* foi de 7 arquivos (mediana = 3, desvio padrão = 13). Estes dados indicam que alterações envolvendo a biblioteca React em projetos JavaScript geralmente alteram poucos arquivos do projeto.

Se tratando das características dos *commits* analisados, observa-se 35 alterações diversas (*change*), 9 correções de problemas (*fix*), 2 *commits* contendo a implementação de novos recursos (*feature*) e 4 refatorações de código. Além disso, observou-se que 9 *commits* tiveram alterações de arquivos que não fossem relacionados à interface gráfica de usuário do projeto. Neste contexto, os 9 *commits* observados compreendem modificações em arquivos de testes. Apresenta-se, na Tabela 5.10, os dados obtidos a partir das análises dos *commits* desta linguagem.

Verifica-se que os dados obtidos com as análises dos projetos JavaScript que utilizam a biblioteca React, apontam para uma tendência à modificação de arquivos de testes quando alterados arquivos que utilizam a biblioteca em questão. Isso vai de encontro ao que observa-se no mercado de aplicações web utilizando JavaScript, já que o uso de testes tem se tornado cada vez mais popular neste contexto.

Tipos de commits	
Fix	9
Change	35
Feature	2
Refactor	4
Release	0
Build	0

Arquivos alterados	
Mínimo	1
Máximo	76
Mediana	3
Média	7

Mensagem com menção à UI	0

Commits com alteração de arquivos não-UI	
Feature	0
Many	0
Backend	0
Tests	9

Commits com alteração de arquivos não-UI	9

Tabela 5.10. Resumo dos *commits* dos projetos JavaScript.

5.2.6 Projetos de Linguagem Python

Em relação aos resultados da linguagem Python, foram minerados os *commits* dos 4 projetos nos quais a biblioteca PyQt foi encontrada. A média foi de 11 arquivos alterados por *commit* (mediana = 5, desvio padrão = 12). Mais uma vez, observa-se que os *commits* envolvendo o uso da biblioteca analisada possuem alterações em poucos arquivos do projeto.

Se tratando das características dos *commits* observados, tem-se 35 *commits* com alterações gerais do código-fonte do projeto (*change*), enquanto 9 *commits* foram correções (*fix*). Além disso, 4 *commits* compreendem refatoração de código (*refactor*) e 2 *commits* com novas funcionalidades (*feature*).

Além disso, 18 *commits* tiveram alterações em arquivos de outras partes do projeto que não fazem parte da interface de usuário. Dentre eles, observou-se 4 *commits* com alterações em arquivos de testes, 3 *commits* com modificações em recursos do *back-end* e 10 *commits* com modificações alguma funcionalidade específica. 6 *commits* possuíram alterações em muitos arquivos distintos do projeto. A Tabela 5.11 sumariza os resultados obtidos com as análises dos 50 *commits* analisados dentre os projetos de linguagem Python.

Observa-se, diante dos dados dos *commits* analisados dentre os projetos Python, que existe pouca relação entre as alterações envolvendo a biblioteca PyQt e arquivos de testes e do *back-end* dos projetos. Por outro lado, 20% dos *commits* analisados possuem modificações em algum recurso específico do projeto. Ademais, 36% dos *commits* tiveram alterações em alguma parte do projeto que não fosse relacionada à interface gráfica de usuário. Dessa maneira, os dados apontam para uma possível relação entre arquivos que utilizam a biblioteca e outros recursos do sistema.

Tipos de commits	
Fix	9
Change	25
Feature	6
Refactor	8
Release	1
Build	1

Arquivos alterados	
Mínimo	1
Máximo	45
Mediana	5
Média	11
Mensagem com menção à UI	1

Commits com alteração de arquivos não-UI	
Feature	10
Many	6
Backend	3
Tests	4
Commits com alteração de arquivos não-UI	18

Tabela 5.11. Resumo dos *commits* dos projetos Python.

A Tabela 5.12 apresenta a sumarização dos resultados obtidos com esta questão de pesquisa.

	Arquivos alterados nos commits				*Características dos commits*	
	Média	**Mediana**	**Mínimo**	**Máximo**	**Arquivos que não são de UI**	**Mensagem com referência à UI**
C Win32	8	3	1	99	14	2
C++ Qt	23	5	1	515	18	8
C# WinForms	6	3	1	119	12	2
Java AWT	71	4	1	2705	21	5
JavaScript React	7	3	1	76	9	0
Python PyQt	11	5	1	45	18	1

Tabela 5.12. Sumarização dos dados obtidos separados por linguagem de programação e biblioteca.

Como as bibliotecas de interface gráfica de usuário se relacionam com as demais partes do código-fonte dos projetos?

Considerando-se as 6 linguagens de programação analisadas, em média 30% dos commits que envolvem as bibliotecas de interface gráfica de usuário, possuem modificações em outras partes dos projetos. Enquanto isso, 70% dos commits alteram somente a interface gráfica de usuário do projeto.

5.3 Implicações

A partir dos resultados obtidos com a execução do presente estudo, possibilita-se observar as implicações daquilo que foi pesquisado.

Se tratando das bibliotecas mais populares observadas, sendo elas Win32 (C), Qt (C++), WindowsForms (C#), AWT (Java), React (JavaScript) e PyQt (Python), observa-se que a popularidade das mesmas se apresenta como um importante indicativo da forte adoção dessas bibliotecas nos

projetos de software livre. Este indicativo, por sua vez, trás implicações para diferentes tipos de perfis de usuários em diferentes contextos. Para os mantenedores das bibliotecas em questão, verifica-se que estão sendo fortemente utilizadas dentre os projetos de suas respectivas linguagens, o que leva a crer que há uma baixa chance de tais bibliotecas caírem em desuso. Além disso, os mantenedores e colaboradores dos projetos de software livre podem se sentir seguros em relação ao uso da biblioteca, uma vez que bibliotecas mais populares possuem alta tendência de continuarem sendo mantidas e atualizadas constantemente. Ressalta-se ainda que, para desenvolvedores que trabalham ou desejam trabalhar com o desenvolvimento de interfaces de usuário, se capacitar no desenvolvimento utilizando tais bibliotecas significa ter uma gama maior de opções de projetos de software livre para contribuir.

Em relação às análises das alterações dos projetos, observou-se que 42% dos *commits* de projetos Java tiveram alterações de arquivos de outras partes do projeto que não fossem interface de usuário. O mesmo aconteceu com 36% dos *commits* de projetos Python e C++, 24% dos *commits* de projetos C# e 18% em relação aos *commits* analisados dos projetos JavaScript. Portanto, os números apresentam um indicativo de que parte considerável das tarefas em projetos de software livre envolvendo alterações em arquivos que utilizam a biblioteca de interface de usuário, também demandem alterações em outras partes do código-fonte dos projetos, como *back-end*, funcionalidades específicas e recursos de modo geral. Isso sugere que colaboradores que contribuem no desenvolvimento das interfaces de usuário desses projetos, precisem ter, em determinados momentos, conhecimento de outras partes do projeto que não somente a interface gráfica de usuário.

Sob uma outra perspectiva, as análises dos *commits* dos projetos mostram que a maior parte dos *commits* analisados possuíram alterações somente em arquivos da interface gráfica de usuário do projeto. Isto é, *commits* envolvendo alterações de arquivos que utilizam a biblioteca de interface gráfica de usuário alteram somente arquivos da interface do projeto. Isso pode ser um facilitador para novos colaboradores, desenvolvedores de *front-end* ou *experts* em alguma das bibliotecas analisadas, que busquem contribuir no desenvolvimento de interfaces de usuário em projetos de software livre. Isto é, na maioria dos casos as tarefas demandarão somente conhecimento dos arquivos que compreendem as interfaces de usuário e não um conhecimento aprofundado em outras funcionalidades, recursos ou regras de negócio do projeto. Vale ressaltar que este indicativo pode ser explorado mais profundamente em pesquisas futuras que tenham como objetivo analisar a relação entre a receptividade de projetos de software livre com tarefas de desenvolvimento de interfaces de usuário dos projetos.

Ainda sobre as análises dos *commits* dos projetos, observou-se que as mensagens de *commits* raramente possuem indicações de que as alterações compreendem mudanças na interface gráfica de usuário do projeto. Esta observação pode ser um indicativo para que os desenvolvedores de projetos de software livre passem a adotar mensagens de *commits* com referência à interface de usuário nesses tipos de alterações, como forma de boas práticas no desenvolvimento dos projetos.

Especificamente em relação aos projetos JavaScript que utilizam a biblioteca React, observa-se um indicativo de que os projetos que utilizam esta biblioteca possuem uma tendência a utilizarem casos de testes. Como já mencionado, tal fato corresponde ao que observa-se no mercado de software da

atualidade. Dessa maneira, este indicativo ressalta a importância dos desenvolvedores e contribuidores em projetos de software livre se capacitarem no desenvolvimento de testes de software, uma vez que podem ser necessários durante o desenvolvimento dos projetos.

5.4 Ameaças à validade

Nesta subseção, serão explicitadas as ameaças à validade tangentes ao método de pesquisa executado neste trabalho. Inicialmente, observa-se que podem existir casos de falsos-negativos na busca pelas bibliotecas. Isto é, casos em que as expressões regulares não tenham identificado o uso de alguma biblioteca em determinado projeto. Como já mencionado, não foi possível monitorar os casos de falsos-negativos no escopo desta pesquisa.

Os resultados apresentados na **QP1** mostram um expressivo número de falsos-positivos durante a busca de determinadas bibliotecas nos projetos. Os maiores casos observados foram as bibliotecas Angular (JavaScript), PyQt (Python), Xamarin (C#), Gtk (C) e wxWidgets (C++). Diante disso, as expressões regulares que identificam tais bibliotecas poderiam ser construídas de forma mais restritiva. Como consequência, teria-se menos casos de falsos-positivos, o que otimizaria o trabalho de análise manual após a busca das expressões. Além disso, a análise manual das ocorrências das bibliotecas nos projetos também pode conter erros devido ao processo exaustivo de analisar as saídas geradas pelo comando `git grep` após a busca pelas expressões regulares.

Ainda sobre as bibliotecas consideradas nesta pesquisa, ressalta-se também que, no escopo deste trabalho, não foram analisadas as licenças de software que tais bibliotecas utilizam. Nesse sentido, é passível de que tenham sido consideradas bibliotecas com licenças restritivas ou de código fechado e que, dessa maneira, não sejam utilizadas em projetos de software livre.

Destaca-se também o processo de seleção dos projetos contendo interface gráfica de usuário, na etapa de Definição do Corpo de Projetos. Neste caso, o processo exaustivo de análise manual pode conter falhas como projetos com interface de usuário sendo descartados, ou mesmo projetos sem interface de usuário sendo incluído no corpo de projetos. Além disso, devido ao processo manual de seleção, definiu-se como padrão selecionar 30 projetos de cada linguagem de programação e, portanto, foi utilizado cálculo amostral para esses casos.

Como mencionado nas análises dos resultados observados, a biblioteca Win32, da linguagem C, possui módulos e componentes relacionados a outros recursos que não são voltados ao desenvolvimento de interfaces gráficas de usuário. Por esse motivo, as análises referentes à essa biblioteca podem conter inconsistências em relação à realidade dos projetos que a utilizam, uma vez que não foi possível analisar os casos individualmente. Sendo assim, não foi identificado se os projetos utilizam a biblioteca para a construção de sua interface de usuário, ou se utilizam outros recursos e módulos providos pela mesma.

Em relação à etapa do estudo que compreende a mineração e análise dos *commits* dos projetos de software livre, mais uma vez o processo exaustivo de análise manual pode representar ameaças à validade daquilo que foi observado. Além disso, a quantidade de *commits* selecionados por

projeto pode representar ameaças que variam de acordo com a amostra selecionada aleatoriamente. Além disso, em pesquisas futuras o processo ideal se faz utilizando uma calculadora amostral para determinar a quantidade de *commits* a serem analisados por linguagem de programação. No escopo deste estudo, isso não aconteceu por conta da limitação de tempo para que a pesquisa fosse executada.

6 CONCLUSÕES

Bibliotecas são componentes de software reutilizáveis. Seu uso é importante no processo de desenvolvimento de projetos de software por contribuir com a economia de tempo e esforço dos desenvolvedores. Em relação às interfaces gráficas de usuário, tem-se que 34% das interfaces desenvolvidas em projetos de software são apoiadas por bibliotecas de componentes de interface gráfica de usuário. Com objetivo de compreender como ocorre a adoção de bibliotecas de interface gráfica de usuário em projetos de software livre, buscou-se, nesta pesquisa, explorar os projetos das linguagens de programação mais populares, analisando estatisticamente a popularidade das bibliotecas e a forma como elas são utilizadas nos projetos.

Com a execução do presente estudo, descobriu-se que as bibliotecas de interface gráfica mais populares dentro do contexto de suas respectivas linguagens foram Win32 (C), Qt (C++), WindowsForms (C#), AWT (Java), React (JavaScript) e PyQt (Python). Além disso, observou-se também que, dentro dos projetos analisados, as alterações em arquivos de interface de usuário que utilizam tais bibliotecas, possuem, em cerca de 30% dos *commits*, alterações em arquivos de outras partes do código-fonte dos projetos. Ainda assim, a maior parte (aproximadamente 70%) dos *commits* observados compreendem alterações somente na interface gráfica de usuário do projeto.

Dentre os projetos C++ que utilizam a biblioteca Qt, concluiu-se que alterações na interface de usuário dos projetos podem demandar conhecimento dos recursos e funcionalidades que o projeto possui. No entanto, observa-se que a maioria das alterações compreendem mudanças somente na interface gráfica de usuário do projeto. Observou-se ainda que nos projetos C# a biblioteca WindowsForms possui baixa relação com outras partes do projeto. Sobre os projetos Java que utilizam a biblioteca AWT, descobriu-se que alterações envolvendo a interface de usuário também demandam alterações em outras partes do projeto. Já se tratando dos projetos JavaScript que utilizam a biblioteca React, os resultados apontam para uma relação entre alterações envolvendo a interface de usuário e uso de casos de teste. Por fim, em relação aos projetos Python que utilizam a biblioteca PyQt, observou-se que existe baixa relação entre a biblioteca e arquivos de testes e do *back-end* dos projetos.

Espera-se, que os resultados gerados com este estudo sejam valiosos para os mais diversos perfis, desde proprietários de projetos de software livre, desenvolvedores, mantenedores das bibliotecas, empresas e para a comunidade científica em geral. Além disso, com esta pesquisa possibilita-se que trabalhos futuros busquem explorar mais profundamente, por meio de uma análise de acoplamento lógico, a relação existente entre os arquivos de interface gráfica de usuário que utilizam as bibliotecas mais populares com as outras partes do código-fonte dos projetos de software livre.

Por fim, a partir do que foi observado durante a execução do presente estudo, ressalta-se a escassez de pesquisas que explorem cenários referentes à utilização de bibliotecas de interface gráfica

de usuário em projetos de software. Nesse sentido, pesquisas futuras se fazem importantes para o desenvolvimento e crescimento dessa área de estudos.

REFERÊNCIAS

BALDASSARRE, Maria Teresa; BIANCHI, Alessandro; CAIVANO, Danilo; VISAGGIO, Giuseppe. An industrial case study on reuse oriented development. In: IEEE. *21st IEEE International Conference on Software Maintenance (ICSM'05)*. [S.l.], 2005. p. 283–292.

BEGEL, Andrew; BOSCH, Jan; STOREY, Margaret-Anne. Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder. *IEEE Software*, IEEE, v. 30, n. 1, p. 52–66, 2013.

BESSEN, James. What good is free software. *Government policy toward open source software*, Washington, DC: Brookings Institution Press, v. 12, p. 27, 2002.

BLOCH, Joshua. How to design a good api and why it matters. In: *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. [S.l.: s.n.], 2006. p. 506–507.

BOGART, Christopher; KÄSTNER, Christian; HERBSLEB, James; THUNG, Ferdian. How to break an api: cost negotiation and community values in three software ecosystems. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. [S.l.: s.n.], 2016. p. 109–120.

BORGES, Hudson; VALENTE, Marco Tulio. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, Elsevier, v. 146, p. 112–129, 2018.

BRITO, Aline; XAVIER, Laerte; HORA, Andre; VALENTE, Marco Tulio. Why and how java developers break apis. In: IEEE. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. [S.l.], 2018. p. 255–265.

CONSTANTINOU, Eleni; AMPATZOGLOU, Apostolos; STAMELOS, Ioannis. Quantifying reuse in oss: A large-scale empirical study. *International Journal of Open Source Software and Processes (IJOSSP)*, IGI Global, v. 5, n. 3, p. 1–19, 2014.

DABBISH, Laura; STUART, Colleen; TSAY, Jason; HERBSLEB, Jim. Social coding in github: transparency and collaboration in an open software repository. In: *Proceedings of the ACM 2012 conference on computer supported cooperative work*. [S.l.: s.n.], 2012. p. 1277–1286.

GOYAL, Raman; FERREIRA, Gabriel; KÄSTNER, Christian; HERBSLEB, James. Identifying unusual commits on github. *Journal of Software: Evolution and Process*, Wiley Online Library, v. 30, n. 1, p. e1893, 2018.

HAEFLIGER, Stefan; KROGH, Georg Von; SPAETH, Sebastian. Code reuse in open source software. *Management science*, INFORMS, v. 54, n. 1, p. 180–193, 2008.

HORA, André; ROBBES, Romain; VALENTE, Marco Tulio; ANQUETIL, Nicolas; ETIEN, Anne; DUCASSE, Stéphane. How do developers react to api evolution? a large-scale empirical study. *Software Quality Journal*, Springer, v. 26, n. 1, p. 161–191, 2018.

IIVARI, Netta; HEDBERG, Henrik; KIRVES, Tanja. Usability in company open source software context-initial findings from an empirical case study. In: SPRINGER. *IFIP International Conference on Open Source Systems*. [S.l.], 2008. p. 359–365.

KALLIAMVAKOU, Eirini; GOUSIOS, Georgios; BLINCOE, Kelly; SINGER, Leif; GERMAN, Daniel M; DAMIAN, Daniela. The promises and perils of mining github. In: *Proceedings of the 11th working conference on mining software repositories*. [S.l.: s.n.], 2014. p. 92–101.

KELTY, Christopher M. Free software/free science. Valauskas, Edward J., 2001.

KIM, Miryung; CAI, Dongxiang; KIM, Sunghun. An empirical investigation into the role of api-level refactorings during software evolution. In: *Proceedings of the 33rd International Conference on Software Engineering*. [S.l.: s.n.], 2011. p. 151–160.

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

LARIOS-VARGAS, Enrique; ANICHE, Maurício; TREUDE, Christoph; BRUNTINK, Magiel; GOUSIOS, Georgios. Selecting third-party libraries: The practitioners' perspective. *arXiv preprint arXiv:2005.12574*, 2020.

LIM, Wayne C. Effects of reuse on quality, productivity, and economics. *IEEE software*, IEEE, v. 11, n. 5, p. 23–30, 1994.

LIMA, Caroline; HORA, Andre. What are the characteristics of popular apis? a large-scale study on java, android, and 165 libraries. *Software Quality Journal*, Springer, v. 28, n. 2, p. 425–458, 2020.

MCILROY, M Douglas; BUXTON, J; NAUR, Peter; RANDELL, Brian. Mass-produced software components. In: *Proceedings of the 1st international conference on software engineering, Garmisch Pattenkirchen, Germany*. [S.l.: s.n.], 1968. p. 88–98.

MILEVA, Yana Momchilova; DALLMEIER, Valentin; BURGER, Martin; ZELLER, Andreas. Mining trends of library usage. In: *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*. [S.l.: s.n.], 2009. p. 57–62.

MILEVA, Yana Momchilova; DALLMEIER, Valentin; ZELLER, Andreas. Mining api popularity. In: SPRINGER. *International Academic and Industrial Conference on Practice and Research Techniques*. [S.l.], 2010. p. 173–180.

MILI, Ali; MILI, Rym; MITTERMEIR, Roland T. A survey of software reuse libraries. *Annals of Software Engineering*, Springer, v. 5, n. 1, p. 349–414, 1998.

MONTANDON, Joao Eduardo; SILVA, Luciana Lourdes; VALENTE, Marco Tulio. Identifying experts in software libraries and frameworks among github users. In: IEEE. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. [S.l.], 2019. p. 276–287.

MORISIO, Maurizio; EZRAN, Michel; TULLY, Colin. Success and failure factors in software reuse. *IEEE Transactions on software engineering*, IEEE, v. 28, n. 4, p. 340–357, 2002.

MORISIO, Maurizio; ROMANO, Daniele; STAMELOS, Ioannis. Quality, productivity, and learning in framework-based development: An exploratory case study. *IEEE Transactions on Software Engineering*, IEEE, v. 28, n. 9, p. 876–888, 2002.

MYERS, Brad; HUDSON, Scott E; PAUSCH, Randy. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*, ACM New York, NY, USA, v. 7, n. 1, p. 3–28, 2000.

MYERS, Brad A. User interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*, ACM New York, NY, USA, v. 2, n. 1, p. 64–103, 1995.

MYERS, Brad A; ROSSON, Mary Beth. Survey on user interface programming. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. [S.l.: s.n.], 1992. p. 195–202.

RAEMAEKERS, Steven; DEURSEN, Arie Van; VISSER, Joost. Measuring software library stability through historical version analysis. In: IEEE. *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. [S.l.], 2012. p. 378–387.

RAEMAEKERS, Steven; DEURSEN, Arie van; VISSER, Joost. An analysis of dependence on third-party libraries in open source and proprietary systems. In: *Sixth International Workshop on Software Quality and Maintainability, SQM*. [S.l.: s.n.], 2012. v. 12, p. 64–67.

ROBILLARD, Martin P. What makes apis hard to learn? answers from developers. *IEEE software*, IEEE, v. 26, n. 6, p. 27–34, 2009.

SCACCHI, Walt; FELLER, Joseph; FITZGERALD, Brian; HISSAM, Scott; LAKHANI, Karim. Understanding free/open source software development processes. *Software Process: Improvement and Practice*, Wiley Online Library, v. 11, n. 2, p. 95–105, 2006.

SOJER, Manuel; HENKEL, Joachim. Code reuse in open source software development: Quantitative evidence, drivers, and impediments. *Journal of the Association for Information Systems*, v. 11, n. 12, p. 868–901, 2010.

SPADINI, Davide; ANICHE, Maurício; BACCHELLI, Alberto. PyDriller: Python framework for mining software repositories. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*. New York, New York, USA: ACM Press, 2018. p. 908–911. ISBN 9781450355735. Disponível em: <http://dl.acm.org/citation.cfm?doid=3236024.3264598>.

STALLMAN, Richard. *Free software, free society: Selected essays of Richard M. Stallman.* [S.l.]: Lulu. com, 2002.

STYLOS, Jeffrey; MYERS, Brad. Mapping the space of api design decisions. In: IEEE. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007).* [S.l.], 2007. p. 50–60.

TIOBE. *TIOBE Index for September 2020.* 2020. Disponível em: <https://www.tiobe.com/tiobe-index/>. Acesso em: 28 de set. de 2020.

WANG, Ying; CHEN, Bihuan; HUANG, Kaifeng; SHI, Bowen; XU, Congying; PENG, Xin; LIU, Yang; WU, Yijian. An empirical study of usages, updates and risks of third-party libraries in java projects. *arXiv preprint arXiv:2002.11028*, 2020.

WU, Ming-Wei; LIN, Ying-Dar. Open source software development: an overview. *Computer*, IEEE, v. 34, n. 6, p. 33–38, 2001.

XAVIER, Laerte; BRITO, Aline; HORA, Andre; VALENTE, Marco Tulio. Historical and impact analysis of api breaking changes: A large-scale study. In: IEEE. *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER).* [S.l.], 2017. p. 138–147.

XIE, Qing; MEMON, Atif M. Model-based testing of community-driven open-source gui applications. In: IEEE. *2006 22nd IEEE International Conference on Software Maintenance.* [S.l.], 2006. p. 145–154.