UNIVERSIDADE FEEVALE

EDUARDO HENRIQUE KASPER

ANÁLISE AUTOMATIZADA DE HABILIDADES DE DESENVOLVEDORES BASEADA EM CONTRIBUIÇÕES NO GITHUB

Novo Hamburgo
2016

EDUARDO HENRIQUE KASPER

ANÁLISE AUTOMATIZADA DE HABILIDADES DE DESENVOLVEDORES BASEADA EM CONTRIBUIÇÕES NO GITHUB

Trabalho de Conclusão de Curso, apresentado como requisito parcial à obtenção do grau de Bacharel em Sistemas de Informação pela Universidade Feevale

Orientadora: Marta Rosecler Bez

Co-orientador: Juliano Varella de Carvalho

Novo Hamburgo
2016

RESUMO

O mercado de trabalho de Tecnologia da Informação (TI) tem mudado muito nos últimos anos, principalmente no que diz respeito ao setor de engenharia e desenvolvimento de *software*. Os grandes investimentos feitos nesta área, decorrentes do avanço sistemático e acelerado de tecnologias, propiciam que mais empresas desenvolvam novas e melhores soluções. Entretanto, devido à vasta sorte de tecnologias, está cada vez mais difícil encontrar profissionais que se adequem às necessidades tecnológicas de empresas. O Github surge, neste contexto, como uma plataforma para desenvolvedores de *software* que possibilita o compartilhamento rápido e fácil de projetos criados por eles. Sendo uma rede social, o sistema gera um volume grande de dados a respeito de seus participantes, como o código escrito e a linguagem de programação empregada. Utilizando a API pública do Github é possível extrair e analisar estes dados de forma a identificar ecossistemas e habilidades que os programadores possuem. Este trabalho dá foco ao desenvolvimento de um protótipo de *software* que atua como um sistema de apoio a decisão na seleção de candidatos, exibindo os ecossistemas com os quais trabalham. Para validar seu funcionamento, é apresentado um experimento realizado com um grupo de 13 pessoas, além de entrevistas com 3 destas para esclarecimentos a respeito dos resultados obtidos. Embora não seja possível identificar todos os ecossistemas que os usuários conhecem devido à falta de publicações ou publicações privadas, é possível identificar ecossistemas que tenham código publicado. No entanto, a hierarquização das linguagens de programação, quando comparada à realidade que cada indivíduo apresenta, não é precisa, em razão da subjetividade de cada um quanto aos seus conhecimentos. Ao final deste trabalho, a análise de todas as respostas comparadas aos resultados provenientes do protótipo é apresentada.

Palavras-chave: Github. Mercado de TI. Análise de Perfis. Seleção e Recrutamento.

ABSTRACT

The market for Information Technology (IT) has changed a lot in recent years, especially regarding the software engineering and development sector. The major investments made in this area, resulting from the systematic and accelerated technology advancements, provide more companies to develop new and better solutions. However, due to the many existing types of technologies, finding professionals who meet companies' technological needs has become increasingly difficult. Github arises, in this context, as a platform for software developers which enables easy and quick sharing of projects created by them. As a social network, Github generates a massive data volume regarding its participants, such as the written code and the used programming language. Github's public API enables extraction and analysis' possibility of this data to identify ecosystems and abilities programmers have. This paper focuses on developing a software prototype which acts as a support system on candidate selection decision, exhibiting ecosystems they work with. An experiment conducted with 13 people, of whom 3 have been interviewed posteriorly, is presented in order to validate its operation and outcome results. Although the identification of all ecosystems users know was not possible due to lack of published code or private contributions, ecosystems which have had published code were found. However, tiering programming languages, in comparison to each individual's reality is not precise because of subjectivity of each in regards to their knowledge. At the end of this study, the analysis of all responses compared to the results from the prototype is presented.

Keywords: Github. IT Market. Profile analysis. Selection and Recruitment.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

ABES	Associação Brasileira de Empresas de Software
ACK	*Amsterdam Compiler Kit*
API	*Application Programming Interface*
Brasscom	Associação Brasileira de Empresas de Tecnologia da Informação e Comunicação
CMS	*Content Management System*
CMT	*Commits*
CSS	*Cascading Style Sheets*
CVS	*Control Version System*
CVCS	*Centralized Version Control System*
DVCS	*Distributed Version Control System*
DI	*Dependency Injection*
GNU	*GNU is Not Unix*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
JSX	*Javascript XML*
IOC	*Inversion of Control*
MVC	*Model-View-Controller*
ORM	*Object Relational Mapper*
PHP	*PHP: Hypertext Preprocessor*
PIP	*Pip Installs Python*
POJO	*Plain Old Java Object*
RCS	*Revision Control System*
REST	*Representational State Transfer*
SCCS	*Source Code Control System*
SCS	*Source Coding Sites*
SPA	*Single Page Application*
TI	Tecnologia da Informação
URL	*Uniform Resource Locator*
VCS	*Version Control System*

SUMÁRIO

INTRODUÇÃO

O advento da TI gerou diversas transformações no mundo contemporâneo, trazendo consigo grandes benesses para a sociedade em geral. Buscar profissionais qualificados tem se apresentado como um grande problema para as empresas, uma vez que as habilidades necessárias para completar projetos de *software* são cada vez mais especializadas e escassas. Uma estimativa realizada pela Associação Brasileira de Empresas de Tecnologia da Informação e Comunicação (Brasscom), divulgada pela Folha de São Paulo, diz que 2014 teve um déficit de 45 mil profissionais (FOLHA DE SÃO PAULO, 2014). Em reportagem de fevereiro de 2016, o Jornal da Globo informou que no Brasil existem 50 mil postos de trabalho que buscam profissionais qualificados (JORNAL DA GLOBO, 2016). É possível, porém, criar maneiras de, utilizando informações disponíveis publicamente, obter mais informações a respeito de prováveis candidatos à vagas disponíveis.

O treinamento de pessoal se enquadra no processo que Chiavenato (2014) chama de "processo de agregar pessoas" (p. 91). Este, segundo o autor, é o processo no qual novas pessoas são agregadas à empresa, e se subdivide em duas etapas: recrutamento e seleção. A etapa de recrutamento, se for externa, busca profissionais fora da empresa; caso seja interna, busca promover ou transferir profissionais que estejam no quadro de pessoal. A segunda etapa de seleção funciona como um filtro sobre candidatos recrutados, selecionando os mais adequados à cada posição (CHIAVENATO, 2014).

O mercado brasileiro de TI, entretanto, não se encontra em situação favorável para recrutamento e seleção de profissionais da área. Segundo a Softex (2014), deverá haver um déficit de, aproximadamente, 408 mil profissionais até 2022. Em 2009, segundo Macedo (2011), o setor de *software* cresceu 4%, mesmo em um contexto de crise, como o de 2008. Matéria da revista Exame (2014) corrobora com os dados, informando que a Associação Brasileira de Empresas de Software (ABES) divulgou pesquisa evidenciando que no ano de 2013 o Brasil obteve um investimento de 15,4% a mais do que em 2012 em desenvolvimento de programas. Em 2014, a ABES apresentou um relatório informando que o setor de *software* cresceu 12,8% sobre 2013.

Neste cenário, é cada vez mais difícil para as empresas encontrarem profissionais com as habilidades necessárias. Segundo matéria publicada pelo portal de notícias Globo

News (2015) empresas estão, até mesmo, treinando profissionais, pois não encontram os perfis desejados no mercado de trabalho. Este fato é corroborado até mesmo no Vale dos Sinos. Conforme entrevistas realizadas por Schaab (2016), empresas buscam treinar profissionais iniciantes por até 6 meses, antes que estes exerçam o ofício para o qual foram contratados. Uma das pessoas entrevistadas por Schaab (2016) afirmou que universidades e escolas técnicas não preparam os profissionais adequadamente.

Entre os segmentos de TI, o desenvolvimento de *software* é o que mais se destaca no Brasil e é preciso entender como estes profissionais são inseridos nas empresas. As inovações tecnológicas e a globalização se mostram como razão de grandes transformações no mercado e, por consequência, em seus atores. Se por um lado esta nova economia traz mais possibilidades por estar em constante desenvolvimento, é preciso que os interessados em trabalhar no setor busquem capacitação constantemente (MACEDO, 2011). Por isto, para se destacar e demonstrar suas competências, desenvolvedores de *software* têm utilizado redes sociais específicas para eles.

Segundo Acioli (2007), redes sociais têm sua origem na área das Ciências Sociais onde se definem como uma conexão de limites variáveis entre atores com vínculos entre si, formando grupos entre suas associações. O Github[1] apresenta-se hoje como uma rede social específica para desenvolvedores, possibilitando o compartilhamento fácil de código entre seus usuários. Este tipo de plataforma permite que programadores criem novos projetos de código aberto e os compartilhem de forma fácil e rápida com qualquer pessoa interessada, utilizando um *software* de controle de versão chamado Git[2] (THUNG; LO; JIANG, 2013). Dabbish et al. (2012) elucidam que um colaborador fica ciente das interações e contribuições de outros colaboradores devido às grandes oportunidades proporcionadas pela plataforma. Pode-se definir, portanto, que o Github, tendo ligações entre seus participantes e possibilitando suas integrações, é uma rede social voltada a colaboração de código de programação.

Nesta rede social são permitidas conexões entre desenvolvedores onde, muitas vezes, centenas deles podem contribuir para um projeto sem se conhecer (THUNG; LO; JIANG, 2013). No sistema, usuários podem ter repositórios (também chamados de projetos) públicos ilimitados. Nestes repositórios o código produzido pode ser disponibilizado a todos, bem

[1] https://github.com/

[2] http://git-scm.com/

como, todas as contribuições feitas a eles. Além disto, qualquer ator integrante pode enviar melhorias de código, fazer comentários, solicitações de correções e alterações, assistir mudanças e demonstrar interesse. Esta facilidade, junto com as ferramentas disponíveis, permitem a disseminação de projetos de código aberto por pessoas e até empresas renomadas, como a NASA, o Google e o Facebook.

Todos estes dados são disponibilizados ao público através de uma API (*Application Programming Interface*) utilizando chamadas REST (*Representational State Transfer*). Neste caso, é possível obter dados sobre seus usuários, como número de contribuições feitas a um projeto, projetos pertencentes à determinados usuários, linguagens de programação utilizadas, conteúdos de projetos, projetos os quais o usuário demonstrou interesse, entre outras informações relevantes e possíveis de serem analisadas para entender melhor o perfil dos mesmos. Dabbish et al. (2012) mencionam que as pessoas tendem a deduzir habilidades técnicas de profissionais da computação baseando-se em suas contribuições no Github. Na verdade, os dados disponíveis já foram analisados por diversos pesquisadores ao redor do mundo. Lima, Rossi e Musoeli (2014) utilizam o projeto Github Archive[3] para analisar e caracterizar contribuições baseadas em relações sociais entre os participantes. Este é apenas um exemplo da utilização da plataforma para os fins mencionados.

Dabbish et al. (2012) fazem importantes aferições a respeito das contribuições de desenvolvedores e a possibilidade de análise de seus proprietários. Segundo os autores, é possível extrair importantes informações do *feed* de atividades de programadores e estimar suas competências técnicas e reputações devido ao modo social no qual a internet tem se tornado nos últimos 15 anos.

Plataformas colaborativas como o Github possibilitam a vasta obtenção e acesso a análises em cima dos dados proporcionados por seus atores integrantes. Aliando este fato ao grande déficit e dificuldades na área de desenvolvimento de *software*, é possível utilizar tecnologias ao alcance da indústria de *software* para facilitar a conexão e a obtenção de informação do contratante sobre o contratado, baseando-se em suas necessidades técnicas. É possível que gerentes de projetos, por exemplo, rastreiem as habilidades de colaboradores dentro da empresa e até mesmo buscar novos colaboradores utilizando estas informações.

---

[3] https://www.githubarchive.org

As referidas necessidades técnicas variam entre as empresas por causa de diversos fatores, como o grande número de linguagens de programação, diferentes *frameworks* utilizados em diferentes projetos, necessidades de clientes, conhecimentos de colaboradores, etc. Muitas vezes, empresas necessitam de profissionais com conhecimento a respeito de tecnologias específicas a cada projeto, e a necessidade de conhecer as capacidades técnicas de desenvolvedores dá-se devido a esta variedade. Assim, as contribuições públicas de usuários no Github apresentam-se como uma forma de facilitar este processo. Por isso, um dos objetivos deste trabalho é desenvolver um algoritmo para a análise de repositórios públicos que identifique em qual(is) ecossistema(s) eles estão inseridos.

Desta forma, este trabalho foca-se em construir e validar um protótipo que possibilite aos responsáveis por administrar times de desenvolvimento a buscar mais informações sobre as habilidades de profissionais, de forma a dar suporte em processos de seleção de candidatos. O trabalho está dividido em 6 capítulos, descritos a seguir.

Capítulo 1 - Desenvolvimento de *software* e codificação social: neste capítulo apresenta-se uma forma de construção de *software* utilizando tecnologias sociais, sistemas de controle de versão, serviços REST e a API de alimentação utilizada pelo protótipo a ser desenvolvido.

Capítulo 2 - Ecossistemas: este capítulo explica quais características devem ser buscadas dos desenvolvedores, como suas habilidades com linguagens de programação e *frameworks*. Ao referenciar os *frameworks*, explica-se o método de seleção e os *frameworks* selecionados.

Capítulo 3 - Protótipo: este capítulo objetiva descrever o protótipo desenvolvido demonstrando seu fluxo de execução para um melhor entendimento do resultado final. Entende-se que, dessa maneira, o entendimento do algoritmo é mais fácil.

Capítulo 4 - Algoritmo: este capítulo explica os processos necessários para a execução da análise de um profissional. Nele, cada procedimento desde a entrada até a saída dos dados é explicado e discutido.

Capítulo 5 - Metodologia: este capítulo explica a metodologia de pesquisa utilizada. Um grupo pré-teste foi utilizado para melhorar os questionamentos e o algoritmo desenvolvido.

Capítulo 6 - Resultados e Discussões: este capítulo exibe e discute os resultados obtidos. Os dados resultantes do experimento são apresentados, junto com a discussão a respeito dos relatórios gerados pelo protótipo. Nele, ainda, as respostas dadas por três entrevistados que foram questionados a respeito de seus resultados são relatadas.

Por fim, as considerações finais são apresentadas, destacando-se o objetivo inicial proposto e a pesquisa realizada, junto à sua conclusão e possibilidades de estudos futuros.

1 DESENVOLVIMENTO DE *SOFTWARE* E CODIFICAÇÃO SOCIAL

De acordo com Acioli (2007), redes sociais originam-se da área das Ciências Sociais, onde se definem como uma conexão de limites variáveis entre atores com vínculos entre si, formando grupos entre suas associações. No meio de TI, recentemente, desenvolvedores de *software* presenciaram o advento de plataformas de codificação social, onde estas apresentam-se como verdadeiras redes sociais, tendo programadores como público-alvo. Nestas redes sociais, desenvolvedores podem divulgar suas atividades e aferir atividades de outros desenvolvedores, deduzindo suas características profissionais (DABBISH et al., 2012; THUNG; LO; JIANG, 2013). Muitas já foram criadas, oferecendo hospedagem de repositórios de códigos, como o Github, o SourceForge[4] e o BitBucket[5].

O fato destas plataformas serem consideradas redes sociais vai ao encontro do que diz Recuero (2009). Segundo ela, redes sociais têm dois elementos: atores (pessoas, organizações; os nós da rede) e suas conexões. Este tipo de plataforma, de acordo com a autora, oferece recursos para que os integrantes construam seus perfis, sendo que as pessoas têm as seguintes motivações para participar das redes: "criar um espaço pessoal; gerar interação social; compartilhar conhecimento; gerar autoridade; e gerar popularidade" (RECUERO, 2009, p. 105). As plataformas para codificação social possuem estas características. De fato, conforme Dabbish et al. (2012), a área de engenharia de *software* pode aproveitar estas tecnologias sociais para melhorar a colaboração e coordenação na construção de programas. O Github, de acordo com eles, provê diversas ferramentas para tal.

O Github tem ganhado muita popularidade nos últimos anos (LIMA; ROSSI; MUSOLESI, 2014). Segundo Sanatinia e Noubir (2016), ele é a maior delas, com mais de 10 milhões de usuários e mais de 16 milhões de repositórios públicos criados. Estas plataformas se baseiam em sistemas de controle de versão, permitindo a criação de projetos, geralmente chamados de repositórios, e o compartilhamento dos mesmos, possibilitando a interação com outros atores. O Github baseia-se em um sistema de controle de versão chamado Git e possui

[4] https://sourceforge.net/

[5] https://bitbucket.org/

diversas ferramentas que incorporam funcionalidades sociais (LIMA; ROSSI; MUSOLESI, 2012).

Tal qual o Github, pode-se citar outras ferramentas que incorporam funcionalidades similares, como o SourceForge, um dos pioneiros neste tipo de serviço, o BitBucket e o GitLab, ferramentas muito similares ao Github e por isso não discutidas especificamente neste trabalho. Neste capítulo aborda-se Sistemas de Controle de Versão e seu histórico, redes sociais existentes e contextualização das mesmas. Após, dá-se ênfase ao serviço oferecido pelo Github para a extração de dados.

1.1. Sistemas de controle de versão

A análise de contribuições no Github só é possível graças ao Sistema de Controle de Versão (*Version Control System* - VCS) utilizado na ferramenta. Segundo Chacon e Straub (2014), VCS são sistemas que gravam as mudanças em arquivos através do tempo, possibilitando a recuperação destas versões. Caso um arquivo seja perdido ou modificado erroneamente, um VCS pode facilmente recuperá-lo. Ruparelia (2010, p. 1) faz um estudo sobre a história dos sistemas de controle de versão, mencionando que VCS são úteis quando mais de um desenvolvedor trabalha no código e, por este motivo, são utilizados na maioria de projetos de *software*.

1.2. Terminologias usadas em sistemas de controle de versão

Alguns termos utilizados por VCS são importantes por contextualizar suas características e formas de usabilidade. Eles podem variar de acordo com o tipo de VCS utilizado, como especificado a seguir. Os termos dissertados no decorrer deste trabalho são:

• **Repositório:** é a peça central de um sistema de controle de versão, armazenando todos os dados (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2011).

• ***Branch*:** declara a revisão predecessora, possibilitando a profissionais responsáveis continuar o desenvolvimento a partir dela (RUPARELIA, 2010).

• ***Commit*:** de acordo com Ruparelia (2010) o *commit* acontece quando um usuário atualiza o repositório com alterações. Loeliger (2009), no entanto, flagra que esta ideia não se aplica ao Git, onde um *commit* grava alterações sem atualizar, naquele instante, o repositório central.

• ***Checkout/Clone*:** segundo Ruparelia (2010) o *checkout* seria o processo onde o repositório é transferido de um servidor para a máquina local. Para o Git, no entanto, este processo tem o nome de *Clone*, como apresentado por Chacon e Straub (2014, p. 44).

• ***Merge*:** o *merge* acontece quando mudanças realizadas por outras revisões são incorporadas à cópia local (RUPARELIA, 2010). Para o Git, diferentemente, como descrito por Loeliger (2010, p. 119), o *merge* acontece quando dois ou mais *branches* são unificados.

1.3. História dos sistemas de controle de versão

É possível afirmar, baseando-se no que diz Loeliger (2009) e corroborado por Ruparelia (2010), que o início da história dos VCS dá-se na primeira metade dos anos 70 e ainda hoje há continuidade na evolução destas ferramentas, devido às diversas contribuições vindas de desenvolvedores ao redor do mundo. Nesta seção contextualiza-se o seu histórico.

1.3.1. SCCS

Segundo Ruparelia (2010), primeiramente desenvolvido na Bell Labs, por Marc J. Rochkind em 1972, o *Source Code Control System* (SCCS) foi o primeiro VCS. Fornecia, de acordo com Loeliger (2009), um repositório e um sistema de chaveamento de arquivos, permitindo a desenvolvedores editarem apenas arquivos destrancados. Quando o profissional finalizava suas modificações, enviava o arquivo novamente, destrancando-o e, assim, permitindo a novos profissionais trabalharem com ele.

O SCCS permaneceu dominante até o lançamento do *Revision Control System* (RCS) (RUPARELIA, 2010). Em seu artigo, entitulado "*The Source Code Control System*" (O Sistema de Controle de Código, 1975, tradução nossa), Rochkind (1975) lista alguns dos maiores problemas com o desenvolvimento de código à época:

• o espaço em disco requerido, para o quê o autor chamou de módulo (hoje comumente conhecido por **revisão** ou **versão)** era muito maior que o necessário para a execução do mesmo;

• correções realizadas a determinadas versões falhavam ao serem aplicadas às outras;

• era complicado referir-se exatamente aos locais de alterações de código;

• caso o cliente tivesse algum problema, seria difícil descobrir sua versão corrente.

1.3.2. RCS

O RCS, desenvolvido em 1980, se diferenciava do SCCS por automatizar o processo de armazenamento, recuperação, *logging*, identificação e combinação (*merge)* de arquivos. Enquanto que, com SCCS, era preciso diversos comandos para realizar as tarefas, o RCS facilitou o processo por utilizar o comando *diff*, comum em ambiente UNIX, para realizar as revisões dos códigos, documentações, e até mesmo suportar arquivos binários, apesar deste suporte ser limitado (RUPARELIA, 2010).

Embora melhor que seu antecessor, o RCS possuía uma grave limitação: não manipulava projetos inteiros ou estruturas de diretórios, apenas arquivos, além de sua sintaxe ser incômoda, o quê dificultava trabalhar em concorrência. Por isso, desenvolvedores não utilizavam seu sistema de *branching*. Ao invés disto, fechavam os arquivos para escrita e trabalhavam apenas em um *branch*. O sistema que facilitou estes processos seria inventado na segunda metade da década de 80, o Sistema de Controle de Versão Concorrente (CVS, na sigla em inglês) (RUPARELIA, 2010).

1.3.3. CVS

Originalmente chamado de CMT, sigla para *commits*, o CVS foi criado em 1985, e posteriormente atualizado em 1986 e 1988, para permitir o trabalho colaborativo para o desenvolvimento de um compilador C, chamado ACK (*Amsterdam Compiler Kit)*. Muitas funcionalidades apresentadas pelo CVS foram seguidas por outros VCS e sua popularidade foi tal que até hoje é possível encontrar-se projetos legados utilizando CVS (RUPARELIA, 2010).

O CVS introduziu novos conceitos para o controle de versão. Utilizava um modelo cliente/servidor, habilitando desenvolvedores de diferentes localidades a funcionar como um time. Cada integrante, assim, trabalhava em sua cópia local e atualiza o servidor conforme necessitava. Este modelo também facilitou a criação de *merges*, incluindo alterações feitas por terceiros nos repositórios locais. Desta forma, não seria mais necessário utilizar um sistema de chaveamento de arquivos, possibilitando o trabalho assíncrono (LOELIGER, 2009). Ruparelia (2010) completa, afirmando que outro grande avanço do CVS foi a possibilidade de manipulação de coleções inteiras de arquivos e diretórios.

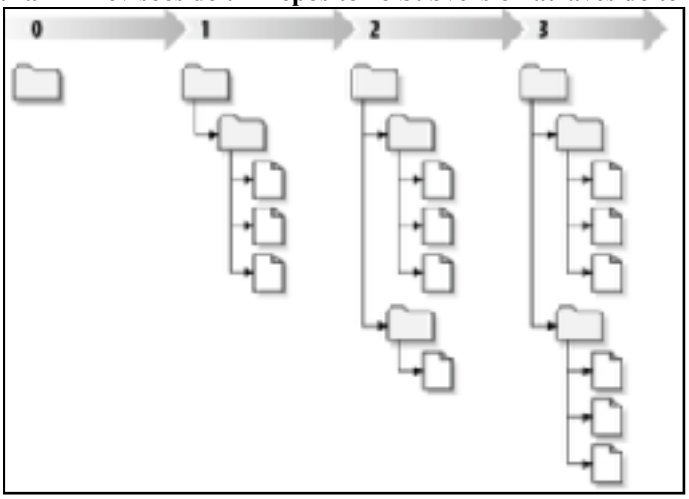
O CVS tornou-se extremamente popular, sendo utilizado em muitos projetos de *software* e substituído pelo Subversion a partir de 2004, o qual programadores se referiam como sendo um "CVS melhor" (RUPARELIA, 2010, p. 7, tradução nossa).

1.3.4. Subversion

O Subversion foi lançado sob licença *open source* e rapidamente adotado pela comunidade, deixando diversas extensões criadas pela mesma. Seu funcionamento, utilizando um modelo cliente/servidor, dá-se através de um formato onde o código é copiado, modificado e unido ao repositório central, prometendo corrigir falhas do CVS (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2011).

No Subversion, o cliente realiza seu *commit*, enviando um número infinito de arquivos e diretórios em uma transação atômica, isto é, transação onde todos os arquivos são enviados ou, em caso de falha, nenhum é. Quando o repositório, localizado no servidor, aceita esta transação, uma nova revisão é gerada[6]. O próximo contribuidor deverá baixar a versão mais recente para, então, modificá-la e enviar a próxima revisão. Na Figura 1 ilustra-se como estas revisões são feitas através do tempo. Porém, o Subversion tem perdido espaço para sistemas de controle de versão distribuídos, como o Git (RUPARELIA, 2010).

Figura 1 - Revisões de um repositório Subversion através do tempo



Fonte: Collins-Sussman, Fitzpatrick e Pilato (2011)

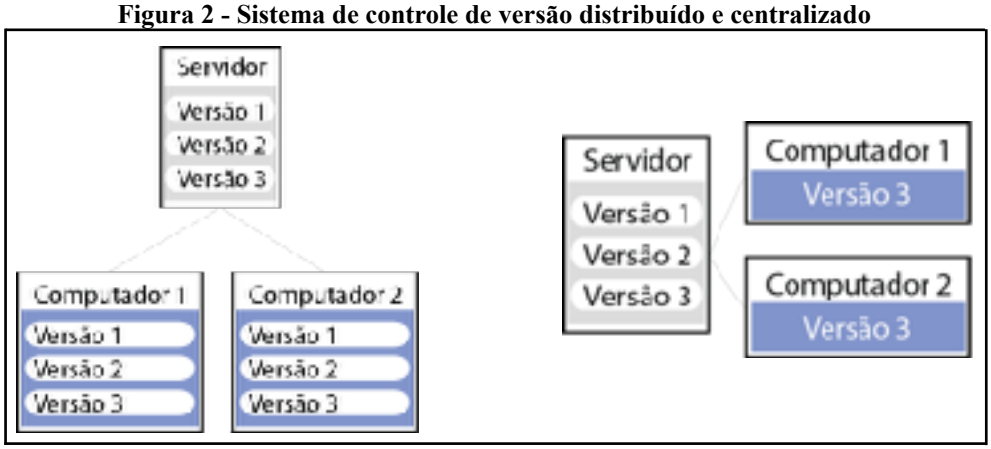
[6] Revisões, no Subversion, são representadas por números naturais e cada nova revisão é incrementada a última. Um repositório vazio, por exemplo, seria representado pela revisão **0**, e a cada novo *commit*, este número seria incrementado (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2011).

1.3.5. Git

De acordo com Ruparelia (2010), o VCS BitKeeper[7], propriedade da empresa BitMover, tinha como sua principal funcionalidade manter cópias locais de repositórios de código e funcionar bem com um repositório central. Em 2002, porém, a proprietária lançou uma licença para BitKeeper, proibindo que a ferramenta fosse utilizada para a criação de ferramentas concorrentes. Apesar disso, muitos desenvolvedores que trabalhavam no Projeto GNU (*GNU is Not Unix*) decidiram utilizá-la.

Como não lhes era permitido ver determinados dados de versões do código para comparar com versões passadas, alguns desenvolvedores começaram a trabalhar em uma ferramenta que suprisse suas necessidades (RUPARELIA, 2010). Em 2005, o projeto Git foi criado por Linus Torvalds para ajudar no desenvolvimento do Kernel do Linux. Logo após, acabou provando-se valioso para outros projetos e rapidamente foi adotado por desenvolvedores Linux (CHACON; STRAUB, 2014).

Figura 2 - Sistema de controle de versão distribuído e centralizado



Fonte: Adaptado de Chacon e Straub (2014)

Um diferencial do Git, segundo Chacon e Straub (2014, p. 29 e 30), frente às outras ferramentas citadas, é que permite copiar todo o sistema de arquivos localmente, e não apenas a última versão de todos os arquivos. Assim, se algum servidor não estiver disponível, uma versão copiada anteriormente pode ser enviada a partir de qualquer cliente para um novo servidor. À este tipo de VCS dá-se o nome de Sistema de Controle de Versão Distribuídos (*Distributed Version Control System* - DVCS). Os demais VCS citados caracterizam-se por ter sempre a última versão localmente, mantendo todo o histórico em um servidor central e, por

[7] http://www.bitkeeper.com

isso, são chamados de Controle de Versão Centralizados (*Centralized Version Control System* - CVCS).

1.4. Plataformas de codificação social

A internet tem se tornado incrivelmente social nos últimos anos. É possível verificar o que as pessoas estão fazendo no Twitter[8], acompanhar suas fotos no Instagram[9] ou conectar-se com elas no Facebook[10] (DABBISH et al., 2012). No meio de engenharia de *software*, os VCS possibilitaram a criação de verdadeiras redes sociais para a codificação colaborativa. Para entender a origem dos dados utilizados pelo protótipo, é preciso que entenda-se o funcionamento da plataforma na qual estão inseridos.

Plataformas de codificação social (*Social Coding Sites* - SCS) são "serviços de mídia social para compartilhamento de projetos na Web" (YOSHIKAWA; IWATA; SAWADA, 2014, p. 1, tradução nossa). Nos SCS desenvolvedores podem começar e desenvolver, interagir e colaborar com outros projetos livremente. Além disso, os sites provêem um ambiente de fácil visualização da atividade de outros profissionais e histórico de projetos hospedados neles.

De acordo com Thung, Lo e Jiang (2013), SCS têm melhorado a colaboração entre desenvolvedores. Sanatinia e Noubir (2016, p. 1, tradução nossa) acrescentam que "tais plataformas facilitam o desenvolvimento ágil e têm o potencial de resolver problemas como a colaboração, comunicação e conflitos de código". Dabbish et al. (2012) fazem um estudo a respeito das interações entre indivíduos. Segundo eles, estas redes sociais podem ser utilizadas para a dedução de habilidades técnicas dos atores que as integram. De acordo com os autores, as pessoas "fazem um rico conjunto de inferências a partir das informações visíveis, como **deduzir objetivos técnicos de alguém baseando-se em ações no código**. Desenvolvedores combinam estas inferências para estratégias efetivas para coordenação de projetos, melhorando suas habilidades técnicas e **gerenciando suas reputações**" (DABBISH et al., 2012, p. 2, tradução nossa, grifo nosso).

---

[8] https://www.twitter.com/

[9] https://www.instagram.com/

[10] https://www.facebook.com/

Nesta seção, faz-se um estudo a respeito do histórico deste tipo de aplicação, a partir da criação do SourceForge, e das funcionalidades oferecidas pelo Github, utilizado para o desenvolvimento do protótipo neste trabalho.

1.4.1. SourceForge

O SourceForge foi o pioneiro em aplicações de hospedagem de repositórios de código, em 1999. Primeiramente, seu foco era hospedar projetos de código aberto (a própria plataforma é hospedada e mantida como um projeto *open source* dentro dela mesma). Em seus primeiros anos, hospedava projetos utilizando CVS e, à medida que o CVS foi sendo substituído pelo Subversion, em 2006, a plataforma passou a hospedar este tipo de repositórios também (MAGUIRE, 2007).

De acordo com Sourceforge (2016), a plataforma possui mais de 3,7 milhões de usuários registrados e mais de 430.000 projetos criados. Algumas das ferramenta providas pela aplicação são:

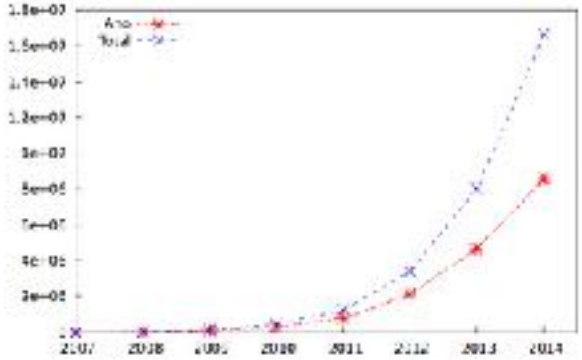
- possibilidade de criar repositórios utilizando os VCS Mercurial, Subversion e Git;
- rastreador de *tickets*, para que seus atores possam controlar *bugs* e funcionalidades a serem corrigidas e desenvolvidas;
- fóruns para discussões;
- páginas para documentação de projetos;
- possibilidade de *download* de projetos e estatísticas sobre os mesmos;
- disponibiliza uma ferramenta para a realização de *merge* entre repositórios.

1.4.2. Github

O Github é um site de codificação social que baseia seu funcionamento em repositórios Git (THUNG; LO; JIANG, 2013). Dabbish et al. (2012, p. 2, tradução nossa) dizem que "[o Github] é um exemplo emblemático de um espaço de trabalho baseado em conhecimento. Este site integra uma série de funcionalidades sociais que tornam visíveis informações únicas sobre seus usuários e suas atividades em projetos de código aberto". De fato, de acordo com Lima, Rossi e Musolesi (2014), o Github não oferece apenas uma plataforma para hospedagem de repositórios, mas uma ferramenta barata (até mesmo gratuita para projetos *open source*) para colaboração de comunidades de desenvolvedores.

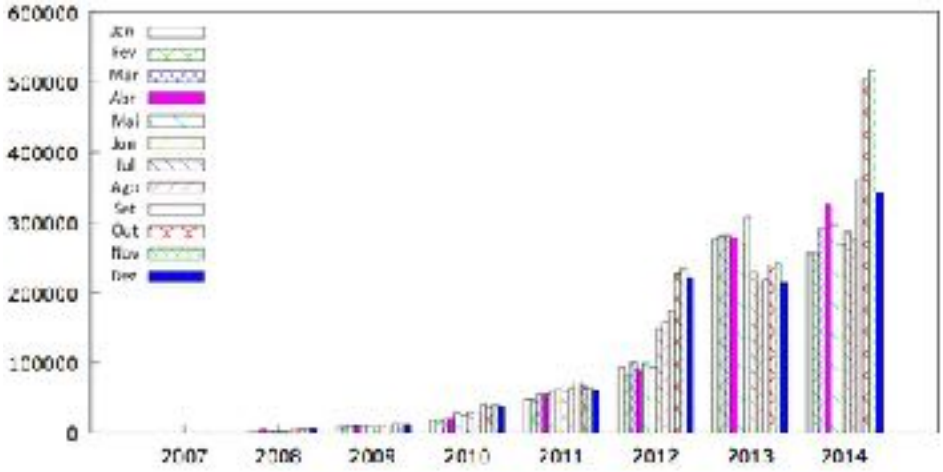
Como apresentado na seção anterior, este tipo de site não é pioneiro neste formato de serviço. Segundo Kalliamvakou et al. (2014, p. 92), porém, o Github introduziu um modelo chamado de "*fork* & *pull*", onde desenvolvedores podem copiar um repositório para sua base (*fork*), modificá-lo e propor esta mudança ao repositório principal (*pull*, também referido como *pull request*). Além disso, criou um modo colaborativo de revisão de código, onde usuários interagem entre si com comentários, mencionando outros atores e apontando trechos de código específicos.

Gráfico 1 - Repositórios criados por ano e total de repositórios criados até 2014



Fonte: Sanatinia e Noubir (2016)

Gráfico 2 - Número de usuários entrantes por ano até 2014

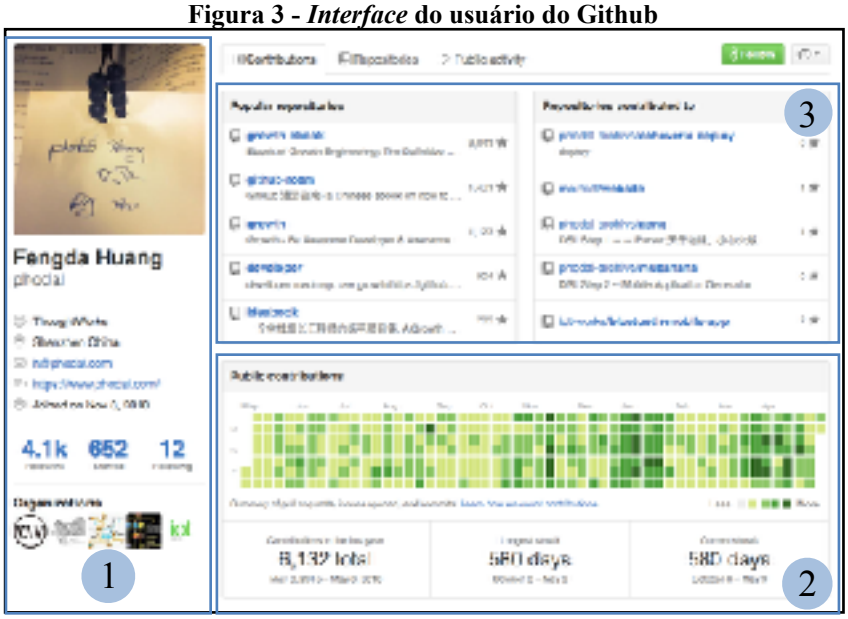


Fonte: Sanatinia e Noubir (2016)

O crescimento do Github tem sido significativo. No Gráfico 1, vê-se a relação entre o total de repositórios e o número de repositórios criados a cada ano, até 2014. É possível verificar que até o fim de 2014 havia mais de 16 milhões de repositórios criados no total, sendo que 8 milhões foram criados apenas naquele ano. Este número tem relação direta com a quantidade de novos usuários na rede. No mesmo período, isto é, entre 2007 e 2014, o número de usuários cresceu exponencialmente, como é possível visualizar no Gráfico 2.

A plataforma se divide em páginas para desenvolvedores e páginas para projetos. Na Figura 3 vê-se um exemplo de página de usuário com seus repositórios. Como página relacionada a um usuário, pode-se citar a do usuário Fengda Huang, cujo usuário é *phodal*[11]. Nesta página é possível encontrar informações sobre o usuário, suas publicações recentes e ainda estatísticas das contribuições públicas do usuário. Segundo Thung, Lo e Jiang (2013), tal transparência é uma funcionalidade importante deste tipo de aplicação.

Figura 3 - *Interface* do usuário do Github



Legenda:
1. Informações públicas, como nome, email, seguidores, empresa para a qual trabalha, etc.
2. Contribuições públicas do usuário.
3. Repositórios populares do usuário e últimos repositórios aos quais o mesmo contribuiu.

Fonte: Github (2016).

Usuários podem, de acordo com Lima, Rossi e Musolesi (2014), realizar as seguintes operações básicas na plataforma:

• criar repositórios Git;

• enviar código para estes repositórios. Cada pessoa que possui permissão para realizar mudanças no repositório é chamada de **colaborador**;

• todo ator do Github pode, também, fazer o *fork* de um repositório, isto é, copiá-lo para sua base, realizar uma nova implementação e propor esta alteração para o repositório principal. Neste caso, os colaboradores vão decidir se aceitam ou rejeitam a modificação. As propostas feitas são chamadas de *pull requests* e, uma vez que este é aceito, esta pessoa passa a ser uma **contribuidora** do repositório. Como mencionado, este modelo foi fundamental

[11] https://github.com/phodal

para a aderência do site entre os programadores e os mantenedores de projetos. Na Figura 4 é possível ver o modelo de *pull requests* implementado, primeiramente, pelo Github;

Figura 4 - Modelo de *pull request* implementado, primeiramente, pelo Github

Fonte: Adaptado de Github (2016)

• seguir pessoas e assistir às suas contribuições e ações na rede;

• usuários também podem **estrelar** um repositório, geralmente utilizado para salvar e verificar depois, ou demonstrar interesse no mesmo. Uma analogia desta funcionalidade com outra bastante conhecida é feita por Sanatinia e Noubir (2016). Seria, de acordo com os autores, similar ao "curtir" do Facebook.

Quadro 1 - Comparação de funcionalidades entre o Github e o SourceForge

	Github	**SourceForge**
Criação de Repositórios	S	S
Rastreabilidade de Tickets	S	S
Documentação	S	S
Pull Requests	S	S
Git	S	S
Subversion	S	S
Mercurial	N	S

Fonte: elaborado pelo autor

No Quadro 1 apresenta-se uma comparação entre os serviços oferecidos pelo SourceForge e o Github. De acordo com o Github Developer (2016), o Github oferece a opção para hospedar repositórios Subversion e uma ferramenta para conversão dos mesmos para repositórios usando Git.

O Github provê uma API RESTful para interação com todos os seus dados. Para citar um exemplo da extensibilidade da API, segundo Lima, Rosi e Musolesi (2014), todos os eventos públicos ocorridos na plataforma, como repositórios copiados, usuários seguidos, repositórios assistidos, repositórios estrelados e propostas de código enviadas, podem ser buscados. Nas próximas seções, trata-se da arquitetura utilizada pelo Github e das especificidades de sua API.

1.5. RESTful *web services*

O REST (*Representational State Transfer*) é "um padrão de arquitetura desenvolvido para sistemas de hipermídia distribuídos, tais como a *World Wide Web*" (SILVA, 2011, p. 28). É comum que *softwares* modernos utilizem este padrão para facilitar a comunicação entre suas partes e com *softwares* de terceiros. O Github é estruturado utilizando este padrão e seus dados são disponibilizados a terceiros. Para buscar os dados dos usuários, o protótipo a ser desenvolvido neste trabalho, realiza requisições REST e, portanto, para melhor entendimento da estrutura utilizada, faz-se necessária a explanação deste modelo.

De acordo com Silva (2011), a ideia deste padrão foi desenvolvida por Roy Fielding, em 2000, durante sua tese de doutorado, enfatizando que o estilo pode ser facilmente escalável. Aplicações desenvolvidas utilizando a arquitetura REST são denominadas RESTful. Por este motivo, serviços *web* estão entre as principais utilizações.

Uma arquitetura REST, de acordo com Fielding (2000), deve conter algumas características, tais como:

• **Cliente-Servidor:** a arquitetura deve separar as responsabilidades através de um modelo cliente-servidor. Deste modo, é possível utilizar diferentes interfaces acessando o mesmo modelo de dados em um servidor, facilitando a escalabilidade de aplicações RESTful.

• **Comunicação sem estado (*stateless*):** uma requisição, de acordo com Fielding (2000), deve ser independente de estados previamente salvos no servidor. Toda informação a respeito do cliente, deve, necessariamente, ser mantida no mesmo. Esta limitação possibilita: visibilidade, pois sistemas de monitoramento não precisam de mais informações além das constantes na requisição para determinar sua natureza; confiabilidade, devido à facilidade de recuperação de falhas parciais; e escalabilidade, uma vez que, não precisando armazenar mais informações entre requisições, recursos são liberados rapidamente. A desvantagem desta abordagem arquitetural, entretanto, se dá devido ao maior tráfego na rede, onde mais dados devem ser enviados a cada nova requisição.

• ***Cache*:** requisições em uma arquitetura REST são passíveis de *cache*. Isto significa que dados enviados ao cliente podem ser armazenados e usados posteriormente em requisições idênticas. Isto possibilita eliminar futuras interações com o servidor.

• **Interface uniforme:** uma interface uniforme de comunicação entre os componentes de uma arquitetura REST mostra-se como sua principal vantagem. Assim, a

visibilidade de interações é melhorada e os elementos são dissociados, permitindo suas evoluções independentes das demais. Por outro lado, a alta padronização da informação pode gerar uma perda de eficiência para casos não ideais.

• **Sistema em camadas:** um sistema em camadas permite que componentes sejam agrupados de forma hierárquica, encapsulando-os e não permitindo que acessem, diretamente, informações a eles não relacionadas. Estes encapsulamentos podem, ainda, ser usados para sistemas legados, utilizando camadas intermediárias para compartilhamento de informação. A desvantagem, entretanto, é a de adição de latência às operações devido ao maior processamento de dados, diminuindo o desempenho.

• **Código sob demanda:** uma arquitetura REST pode proporcionar ao cliente o *download* de extensões na forma de *applets* ou *scripts*, aumentando sua extensibilidade e reduzindo o número de implementações necessárias no lado do cliente, e, consequentemente, simplificando-o. Porém, como esta característica reduz a visibilidade do sistema, ela é considerada **opcional**.

Conforme Silva (2011), as interações entre o cliente e o servidor devem ser mediadas por meio de métodos HTTP (*Hypertext Transfer Protocol*), que, por sua vez, devem expor recursos do servidor. Os métodos HTTP são:

GET - Uma requisição *GET* é utilizada para recuperar um recurso. Após o envio, o cliente recebe um conjunto de cabeçalhos e a representação do mesmo (SILVA, 2011; RICHARDSON; RUBY, 2007, p. 79). Esta é a requisição principal utilizada pelo protótipo, uma vez que precisa-se recuperar recursos, como usuários, repositórios, arquivos dos repositórios, contribuições do usuário, etc.

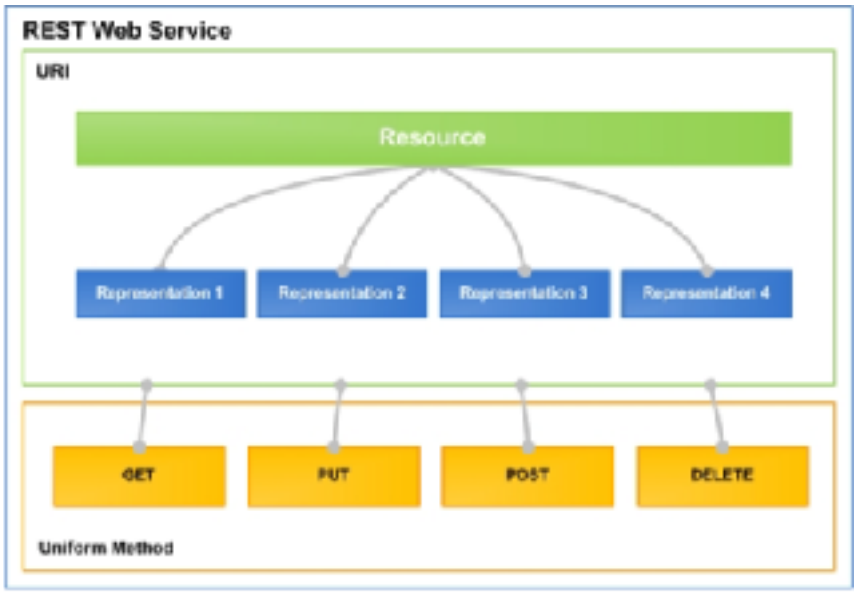
POST - Uma requisição *POST* é utilizada para tentar criar um novo recurso no servidor, enviando uma nova representação (SILVA, 2011).

PUT/PATCH - Uma requisição *PUT* asserta que um recurso deve ser atualizado. Nela, um novo corpo para o recurso é enviado ao servidor para que este a modifique (RICHARDSON; RUBY, 2011).

DELETE - Uma requisição *DELETE* apaga um recurso. O retorno desta solicitação pode ser um *status*, ou nada (SILVA, 2011; RICHARDSON; RUBY, 2007, p. 79).

Na Figura 5 representa-se as interações em uma arquitetura REST e os dados passíveis de modificação no servidor.

Figura 5 - Diagrama de um serviço *web* RESTful



Fonte: Silva (2012, p. 33)

1.6. A API do Github

As informações armazenadas nos servidores do Github são disponibilizadas através de sua API. Segundo Sanatinia e Noubir (2016, p. 2, tradução nossa), "o Github provê uma API RESTful (atualmente na versão 3) para fazer consultas sobre usuários e repositórios", retornando as informações paginadas no formato *JavaScript Object Notation* (JSON)[12].

De acordo com o site Github Developer (2016), a API possui uma limitação de 5000 requisições feitas por hora para usuários autenticados e 60 requisições para usuários não autenticados. O portal de desenvolvedores também explica algumas formas de autenticação de usuários. No caso deste estudo, utiliza-se um *token* (que pode ser gerado a partir de qualquer conta no Github) em cada requisição feita. Desse modo, a API mantém um rastreamento do número de requisições já feitas.

Por ser uma API RESTful, as requisições feitas são familiares, uma vez que utilizam os verbos REST supracitados. No Quadro 2 é possível visualizar os métodos HTTP e suas respectivas descrições. Todas as buscas, como usuários, repositórios e arquivos dos repositórios dão-se através de requisições GET e retornam, invariavelmente, objetos JSON (GITHUB DEVELOPER, 2016).

[12] Um formato leve para troca de dados que facilita a leitura e escrita por humanos, e a escrita e o processamento por computadores. Fonte: http://www.json.org, acesso em 9 de abr. de 2016.

Quadro 2 - Verbos HTTP possíveis de serem utilizados com a API do Github

Verbo	Descrição
HEAD	Pode ser usado contra qualquer recurso, apenas para recuperar o cabeçalho HTTP
GET	Usado para recuperar um recurso
POST	Usado para criar recursos
PATCH	Usado para atualizar algum recurso parcialmente
PUT	Usado para substituição de recursos ou coleções
DELETE	Usado para exclusão de recursos

Fonte: Github Developer (2016)

Para buscar um usuário, a requisição feita é **GET /users/:username**, onde **:username** é uma variável que representa o nome do usuário. Uma requisição GET em /users/phodal[13], por exemplo, retornará um objeto JSON com informações públicas a respeito deste usuário, como visto na Figura 6. Da mesma maneira, informações a respeito de um repositório específico podem ser acessadas a partir de uma requisição **GET /repos/:owner/:repo**, onde **:owner** e **:repo** são variáveis representando o proprietário e o repositório, respectivamente. Por exemplo, informações a respeito do repositório **Hello-World**, do usuário **octocat**, podem ser buscadas acessando **/repos/octocat/Hello-World**[14] (GITHUB DEVELOPER, 2016).

[13] Esta requisição pode ser visualizada no navegador, acessando https://api.github.com/users/phodal

[14] O retorno desta requisição, omitida neste trabalho devido à sua extensibilidade, pode ser visualizada acessando a URL https://api.github.com/repos/octocat/Hello-World

Figura 6 - Exemplo de retorno em formato JSON, buscado da API do Github

Fonte: Github Developer (2016)

1.7. Resumo do capítulo

Este capítulo é utilizado como base teórica para a apresentação dos demais conceitos abordados neste trabalho. Nele, avaliou-se os sistemas de controle de versão, seu funcionamento e histórico. Isto é importante para que se entenda a tecnologia na qual as plataformas de codificação social são baseadas.

Estas plataformas apresentam-se hoje como verdadeiras redes sociais, possibilitando o desenvolvimento de *software* em conjunto com desenvolvedores de todas as partes do mundo. A ênfase principal foi no Github, pois é a aplicação foco deste estudo. Sobre ele, explicou-se a respeito de seu funcionamento, características e o meio de extração de dados.

A extração de dados dá-se através da API RESTful do Github, motivo pelo qual, apresentou-se o histórico da arquitetura REST e as características as quais *softwares* são arquitetados desta maneira. O Github, um *software* assim concebido, disponibiliza seus dados através de requisições GET feitas à sua API. Alguns exemplos destas requisições também foram apresentados.

Com isto, o objetivo específico **“Explorar a API do Github”** foi atendido, uma vez que define-se o modo como os dados são extraídos. No próximo capítulo, define-se a utilização de ecossistemas neste trabalho e faz-se um estudo a respeito dos ecossistemas analisados pelo protótipo.

2 ECOSSISTEMAS

Veículos midiáticos, como a Folha de São Paulo (2014) e o Jornal da Globo (2016), afirmam que empresas de TI, no Brasil, não pararam de contratar mesmo em ambiente de crise. Além disso, a Folha de São Paulo (2014) prevê um déficit de 45 mil profissionais do setor para os próximos anos, conforme dados da Brasscom (Associação Brasileira de Empresas de Tecnologia da Informação e Comunicação). Neste cenário, as capacidades técnicas dos profissionais do mercado se mostram como principal empecilho para suas contratações, pois, conforme demonstrado por reportagem da Globo News (2015), empresas estão treinando seus profissionais para suprir suas vagas. Este fato também é citado por Schaab (2016). Segundo ele, existem empresas que, além de capacitações técnicas pontuais, treinam seus profissionais iniciantes por até 6 meses, pois as universidades e as escolas técnicas não conseguem prepará-los adequadamente.

Neste contexto, a principal análise a ser feita pelo protótipo aqui proposto é a de identificação dos ecossistemas, linguagens de programação e *frameworks*, aos quais os atores analisados estão familiarizados. Isto é importante, pois, de acordo com Sanatinia e Noubir (2016), muitos artefatos modernos de *software* utilizam mais de uma linguagem de programação para atingir diferentes objetivos. O Github surge, desta forma, devido à sua API e o vasto volume de dados a respeito de seus usuários integrantes, como um canal onde seja possível a realização desta análise. A partir desta ferramenta será possível extrair os ecossistemas dos usuários e auxiliar na análise a respeito dos desenvolvedores de *software*.

No decorrer deste trabalho, o termo **ecossistema** é utilizado para descrever o contexto no qual programadores estão inseridos em termos tecnológicos. Ou seja, as tecnologias de sua *expertise*, como linguagens de programação com a qual trabalham e *frameworks* que usam rotineiramente. Neste capítulo, contextualiza-se e define-se os ecossistemas a serem analisados.

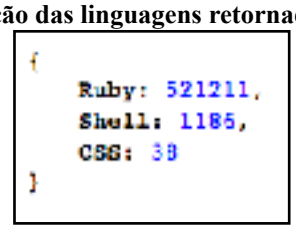
2.1. Linguagens de programação

Conforme Sebesta (2011), linguagens de programação têm sido utilizadas para diversos fins (desde aplicações empresariais, jogos eletrônicos até controle de usinas nucleares), e por isso, há uma grande diversidade entre as linguagens existentes.

Programadores que dominam mais de uma linguagem, de acordo com o autor, desenvolvem maior capacidade de expressão de ideias, melhor embasamento para escolhas mais adequadas em projetos, maior habilidade de aprendizado de novas linguagens, melhor entendimento da implementação de código e aperfeiçoamento de uso das linguagens já existentes. É essencial para a análise de perfil de um desenvolvedor, entender quais linguagens de programação o mesmo domina. No algoritmo desenvolvido durante este trabalho, busca-se saber, não apenas as linguagens familiares ao profissional analisado, mas também os *frameworks* conhecidos.

Segundo o Github Developer (2016), a API da plataforma possui a funcionalidade de retornar as linguagens inclusas no repositório. A informação pode ser obtida através de um GET em **/repos/:owner/:repo/languages**, onde **:owner** e **:repo** são variáveis para o proprietário e o repositório, respectivamente. A Figura 7 apresenta um exemplo do retorno de uma requisição feita no repositório **octokit.rb** do usuário **octokit**[15]. O número à direita representa a quantidade de *bytes* relativos àquela linguagem presente no repositório em questão.

Figura 7 - Representação das linguagens retornadas pela API do Github



Fonte: Github Developer (2016)

Prabhakar, Litecky e Arnett (2005) identificaram, em seu artigo, as áreas com demandas mais altas de profissionais no mercado de TI em 2005. Os autores apontam que, naquele ano, o desenvolvimento para internet se mostrava como a área mais promissora, e as linguagens de programação para este meio despontavam como as mais pedidas por empresas. Eles concluem dizendo que: “em um mercado de trabalho de melhoramento contínuo, é cada vez mais importante para os profissionais de TI estarem cientes da demanda por habilidades específicas” (PRABHAKAR; LITECKY; ARNETT, 2005, p. 94, tradução nossa).

Como pode ser inferido da conclusão de Prabhakar, Litecky e Arnett (2005), uma das principais características a ser analisada de um desenvolvedor de *software* são as linguagens de programação com as quais trabalha. Por esta razão, o protótipo busca na API do Github informações a respeito de linguagens que o perfil analisado possivelmente conhece. Deste

[15] Esta requisição pode ser visualizada acessando: https://api.github.com/repos/octokit/octokit.rb/languages

modo, durante um processo de seleção para contratação, ou alocação em projetos de *software*, é possível visualizar habilidades técnicas na linguagem necessária.

2.2. Frameworks

Frameworks são ferramentas importantes para o desenvolvimento de aplicações modernas. De acordo com Johnson e Foote (1988), *frameworks* são úteis para reusar mais que apenas as aplicações de código usuais. Eles permitem a programadores estender suas bibliotecas (*libraries*) para uso personalizável. Assim, esta abordagem garante a reutilização e consistência dos componentes após mudanças na implementaçã o. Os autores também apontam que, devido ao reuso de suas funcionalidades, não é supresa que desenvolver *frameworks* seja uma tarefa que requer experiência e experimentação.

Note-se, porém, que Johnson e Foote (1988) diferenciam *frameworks* e bibliotecas. Segundo eles, os *frameworks* são mais que apenas bibliotecas bem escritas. As bibliotecas servem como peças discretas, isto é, cada uma corresponde à funcionalidades para resolver problemas diversos. Por exemplo, de acordo com os autores, pode-se citar classes que manipulam coleções em Smalltalk, como *Arrays*, *Dictionaries*, *Sets* e *Bags*. Os *frameworks*, de outra forma, geralmente são formados de diversas bibliotecas, estendendo, às vezes, outros *frameworks*, cuja implementação serve para propósitos específicos.

Johnson e Foote (1988) citam a inversão de controle como a principal característica deste tipo de *software*. Segundo eles, métodos definidos pelo usuário são chamados pela aplicação, ao invés dessa chamada ser feita diretamente pelo usuário: "o *framework* geralmente faz o papel de programa principal na coordenação e sequenciamento das atividades da aplicação. Esta **inversão de controle** lhes dá o poder de servir como esqueletos extensíveis" (JOHNSON; FOOTE, 1988, tradução nossa, grifo nosso). Assim, o código escrito pelo usuário é executado junto ao algoritmo fornecido.

A inversão de controle também é citada por Fowler (2005) como a principal característica ao diferenciar *frameworks* e bibliotecas. As bibliotecas, de acordo com o autor, são geralmente organizadas em classes, que podem ser chamadas de forma sequencial, controladas pelo usuário. O *framework*, por outro lado, inclui um sistema abstrato com comportamentos pré-definidos que fazem chamadas ao código passado pelo usuário. De

acordo com Fowler (2005), é o chamado princípio de Hollywood: “não nos chame, nós chamaremos você” (tradução nossa[16]).

Devido à esta definição, a análise de perfis dos desenvolvedores foca-se especificamente nos *frameworks* que desenvolvedores utilizam e que fazem parte de sua *expertise*. Não faria sentido analisar bibliotecas às quais eles estão acostumados a utilizar, uma vez que seus códigos podem utilizar diversas bibliotecas para as mais variadas finalidades. Para analisar *frameworks*, no entanto, é preciso definir o método de escolha dos mesmos.

2.2.1. Método de seleção de *frameworks*

Nesta seção, aborda-se o método escolhido para seleção dos *frameworks*. É importante que esta seleção seja feita de forma coerente com os objetos analisados e o meio em que estão inseridos. Isto é, uma vez que se queira analisar o perfil de desenvolvedores que estão inseridos no Github, pelos motivos expostos no Capítulo 1, é lógico que se utilize da mesma plataforma para a seleção dos *frameworks*.

Em um primeiro momento, cogitou-se pelo Tiobe Index[17]. Desde 2001, o Tiobe Index oferece uma lista das linguagens mais populares no momento, gerada por seu algoritmo proprietário. A busca é feita através de motores de busca, como o Google[18] e Yahoo[19], além de sites que possuem conteúdo relacionado às linguagens, como Youtube[20] e a Wikipedia[21]. Baseando-se no volume de dados coletados, o Tiobe Index utiliza seu algoritmo para ranquear as linguagens de programação mais populares (TIOBE INDEX, 2016).

O Github também permite ranquear linguagens de programação utilizando-se do serviço para busca de repositórios[22]. Neste serviço, é possível inserir consultas específicas para filtros dos dados por diversos campos, como data de criação do repositório, linguagem de programação principal, palavras-chave, etc. Além destas buscas, é possível filtrar e ordenar

---

[16] “*Don’t call us, we’ll call you”* (FOWLER, 2005).

[17] http://www.tiobe.com/tiobe_index

[18] https://google.com.br

[19] https://br.yahoo.com/

[20] https://www.youtube.com/

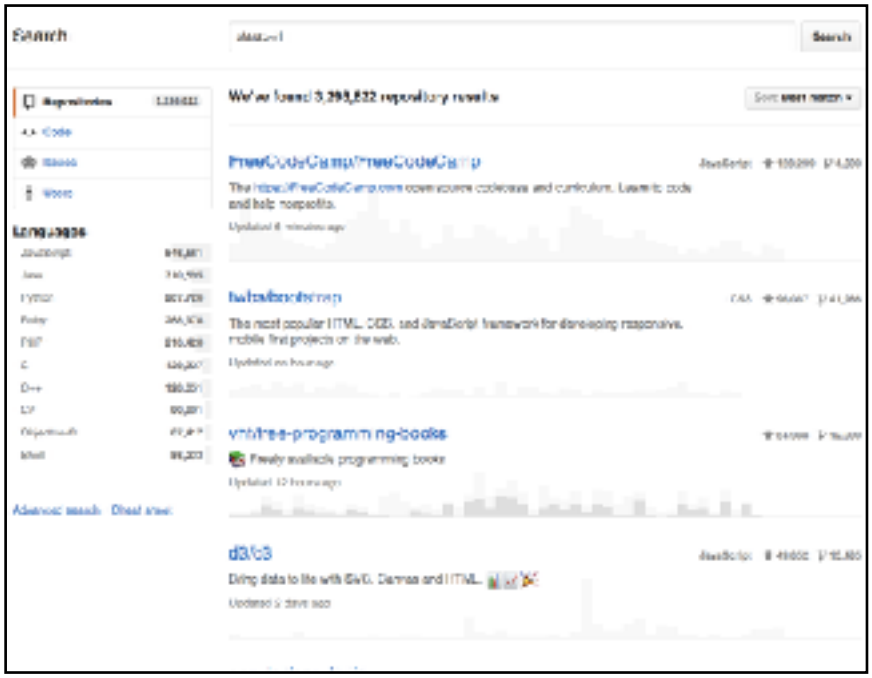
[21] http://wikipedia.org

[22] https://github.com/search

repositórios pelo número de estrelas que estes recebem de usuários (GITHUB HELP, 2016). Como visto no Capítulo 1, o Github oferece a opção de **estrelar** repositórios, ferramenta similar ao **curtir** do Facebook, indicando interesse no repositório por parte do usuário.

Para buscar os repositórios mais estrelados, insere-se a consulta **stars:>=1**[23]. Juntamente com os repositórios mais estrelados, a plataforma retorna as linguagens de programação principais utilizadas, conforme ilustrado pela Figura 8. Dessa forma, é possível obter a lista de linguagens de programação e repositórios mais populares no Github.

Figura 8 - Retorno da busca de repositórios mais estrelados



Fonte: Github (2016)

Uma vez que o protótipo utiliza dados vindos do Github, como usuários e seus repositórios, optou-se por selecionar as linguagens de programação e *frameworks* da mesma maneira. Isto é, a partir da procura dos repositórios mais populares, busca-se, entre os repositórios retornados pelo Github, os *frameworks* mais populares. Como amostra deste estudo, definiu-se arbitrariamente o número de dez *frameworks* para análise pelo protótipo, a partir da definição das cinco linguagens mais populares no Github. Com base no resultado exposto, busca-se os *frameworks* a serem analisados e a seleção final dá-se apoiada nos repositórios mais estrelados.

[23] https://github.com/search?q=stars%3A%3E%3D1

As linguagens selecionadas, como é possível verificar na Figura 8, à esquerda, são: JavaScript, Java, Python, Ruby e PHP. Para cada uma destas linguagens, então, é preciso selecionar seus *frameworks* mais populares.

2.3. *Frameworks* analisados

Nesta seção, descreve-se brevemente os *frameworks* selecionados a partir da busca do Github. Como os repositórios retornados pela busca são diversificados, pois são compostos por projetos, bibliotecas, conteúdo para aprendizado, entre outros, buscou-se pelos projetos que se autodenominam *frameworks* e atendem à definição de inversão de controle descritos anteriormente. Estes *frameworks* têm, por característica principal, facilitar o trabalho de desenvolvedores.

2.3.1. JavaScript - AngularJS

Aplicações *web* modernas tornaram-se maiores e mais rápidas a usuários, e mais complexas para desenvolvedores. Devido à esta complexidade, começou-se a utilização de *frameworks* para auxiliar na velocidade, testabilidade e manutenção de aplicações. O AngularJS foi criado para estas necessidades. O AngularJS oferece uma estrutura básica, utilizando padrão MVC[24] (*Model-View-Controller*) (SESHADRI; GREEN, 2014).

Aplicações desenvolvidas utilizando AngularJS, segundo Seshadri e Green (2014), possuem algumas características e benefícios, citados a seguir:

- Aplicações desenvolvidas com AngularJS são chamadas SPA (*Single Page Application* - Aplicações de Uma Página);
- Uma aplicação AngularJS necessita de menos linhas de código para completar tarefas se comparada à mesma aplicação utilizando outras soluções, como JavaScript puro usando jQuery[25];
- Por causa da estrutura básica oferecida pelo *framework*, desenvolvedores podem focar-se na lógica de negócios ao invés de funcionalidades básicas;

---

[24] Segundo Seshadri e Green (2014), MVC é um padrão de arquitetura de *software* que separa logicamente o *Model* (entidade responsável pelos dados), o *View* (entidade responsável pela visão e interação do usuário) e o *Controller* (entidade responsável pelas ações entre *Models* e *Views*). Este padrão é utilizado para separar as responsabilidades de cada entidade, tornando-as independentes e facilitando a criação de novas funcionalidades.

[25] http://jquery.com

- *Templates* em AngularJS são escritos utilizando HTML puro;
- Pacotes de terceiros podem ser inseridos facilmente em uma aplicação AngularJS.

Este *framework,* desenvolvido pela Google é, de fato, o mais popular entre os repositórios do Github. Os motivos anteriormente expostos podem explicar a razão desta popularidade. Na próxima seção, aborda-se outro *framework* similar, o React.

2.3.2. JavaScript - React

O React é um *framework* criado pelo Facebook para abstração de componentes HTML (VEPSÄLÄINEN, 2016). Segundo o Facebook Inc. (2016), é considerado a *View*, em um padrão MVC, pois o React gerencia alterações em documentos automaticamente dependendo de alterações nos dados. Isso o torna simples para a construção de grandes interfaces.

O React funciona utilizando uma árvore de documento virtual, isto é, um elemento React é uma representação virtual de um elemento HTML. Desta forma, quando uma mudança de dados acontece, o algoritmo responsável por alterar, compara os elementos virtuais e ajusta apenas os componentes necessários, aumentando a performance. Para fazer isto, o *framework* possui uma API própria para o desenvolvimento de componentes, o JSX (*JavaScript XML*), cuja sintaxe se assemelha muito com a linguagem HTML (VEPSÄLÄINEN, 2016).

React definitivamente não é um *framework* como AngularJS, por não possuir diversas funcionalidades nativas (VEPSÄLÄINEN, 2016). Ainda, alguns autores definem React como uma biblioteca para construção de interfaces, como é o caso de Vepsäläinen (2016). Neste trabalho, porém, devido ao conceito de *framework* apresentado, considera-se React um *framework* (programadores inserem ações a serem tomadas, e a ferramenta decide quando realizá-las).

2.3.3. PHP - Laravel

Saunier (2014) diz que, de todas as linguagens para programação de aplicações no lado do servidor, PHP tem a mais frágil “barreira de entrada” (SAUNIER, 2014, p. 8, tradução nossa). Segundo o autor, este fenômeno dá-se devido à facilidade de adicionar funcionalidades da linguagem aos documentos HTML, o que a tornaria, em tese, de fácil

aprendizagem por usuários que saibam diagramar páginas *web*. Entretanto, o autor aponta que esta facilidade acaba complicando o código proporcionalmente à complexidade da aplicação. Laravel, então, seria o *framework* que permitiria a facilidade na estruturação e reuso do código.

Segundo Saunier (2014), algumas das características do *framework* são:

• **Modularidade:** é composto de mais de 20 bibliotecas diferentes, dividindo-as em módulos integradas utilizando um *software* gerenciador de dependências, o Composer[26].

• **Testabilidade**: possui diversas funções e métodos para auxiliar a testabilidade do código.

• **Roteamento:** o *framework* permite a programadores definir facilmente suas rotas, a partir dos métodos REST (*GET*, *POST*, *PUT* e *DELETE*).

• **Mapeamento Relacional de Objetos (*Object Relational Mapper* - ORM)**: possui nativamente a funcionalidade de construir consultas ao banco de dados automaticamente, ao invés de o programador precisar escrevê-las manualmente, chamada **Eloquent**. Utilizando o Eloquent, pode-se buscar dados de diferentes bases de dados (PostgreSQL, SQLite, MySQL e SQL Server) utilizando a mesma sintaxe.

O *framework* também possui funcionalidades para envio de emails, gerenciador de *template*, autenticação de usuários, integração a serviços de filas, entre outros. Segundo Saunier (2014), as principais características do Laravel são sua simplicidade e expressividade. Por isto, é possível realizar ações como se fosse em inglês simples.

2.3.4. PHP - Symfony

O Symfony é um *framework* PHP focado em reduzir o tempo de desenvolvimento de aplicações sem sacrificar "a manutenibilidade, a escalabilidade ou a qualidade" (BOWLER; BANCER, 2009, p. 7, tradução nossa). O Symfony utiliza o padrão MVC para separar as responsabilidades de cada classe e garantir a divisão da lógica de apresentação e de negócios (BOWLER; BANCER, 2009).

Segundo Bowler e Bancer (2009), algumas das principais características do Symfony são:

---

[26] https://getcomposer.org/

• **Formulários e Validações**: o *software* divide os formulários em dois tipos: **Propel** e **Simples**. Formulários **Propel** são aqueles cujas validações estão vinculadas diretamente à regra no banco de dados, sendo customizáveis pelo desenvolvedor. Os formulários **Simples**, apesar de poderem ser validados, não persistem suas informações em um banco de dados.

• ***Plugins***: *plugins* são "uma das melhores funcionalidades da arquitetura do Symfony" (BOWLER; BANCER, 2009, p. 10, tradução nossa). Muitos artefatos de *software* podem ser escritos e integrados ao *framework*, permitindo assim, serem reutilizados inúmeras vezes. Entre os *plugins* mais populares apresentam-se funcionalidades para gerenciar arquivos estáticos *web* (arquivos CSS, imagens, arquivos JavaScript), funcionalidade para construção fácil de blogues e para a criação de CMS (*Content Management System*).

• **Internacionalização**: o Symfony oferece *interfaces*, padrões e implementações de localizações para auxiliar a internacionalização e localização.

• **Geradores**: quando se está escrevendo uma aplicação *web*, o administrador geralmente precisa de uma área onde ele gerencia o conteúdo do site. O Symfony oferece geradores de estruturas pré-prontas, que criam formulários e listagens a partir da linha de comando, baseando-se no conteúdo dos *models*.

Além dos itens citados, o Symfony oferece implementações para *cache*, testes unitários e customizações no *framework* (BOWLER; BANCER, 2009). Aparentemente, é um *software* sólido, principalmente pela sua extensibilidade através de *plugins.* Portanto, para a linguagem de programação PHP, o protótipo deverá ser capaz de detectar tanto o Symfony, quando o Laravel.

2.3.5. Ruby - Rails

Publicamente lançado em 2004, conforme Geer (2006), Rails é um *framework* para o desenvolvimento de aplicações *webs* desenvolvido para ser usado pela empresa do seu criador e, mais tarde, lançado a público. É focado principalmente em aplicações que necessitam de banco de dados e utilizam o padrão MVC. Aplicações Rails seguem um princípio de "convenção acima de configuração" (GEER, 2006, p. 18, tradução nossa).

Uma aplicação desenvolvida com Rails, de acordo com Geer (2006), tem em seu ORM (*Object Relational Mapper*) a habilidade de mapeamento não apenas de colunas de tabelas do banco de dados, mas também seus relacionamentos. Isto se dá devido ao princípio

de convenção ao invés de configuração, ou seja, ao invés de configurar como o *framework* deve funcionar, Rails utiliza um padrão de estrutura e nomenclatura pronto, e seguido pelos desenvolvedores. Desta forma, programadores Rails não precisam especificar suas *queries* para modificações e seleções no banco de dados.

Como vantagem do desenvolvimento com Ruby e Rails, Geer (2006) cita a velocidade. Ele afirma que um desenvolvedor Rails pode fazer o trabalho de um time inteiro usando Java, e demorar um mês em um projeto que, utilizando Java, demoraria seis meses. Além disto, o autor informa que Rails pode desenvolver todas as partes de uma aplicação de forma eficiente, devido ao seu outro princípio de "não repetir a si mesmo" (GEER, 2006, p. 19, tradução nossa). O *framework*, de acordo com o seu criador, faz suposições: "Rails diz 'Vamos simplificar para o quê a maioria das pessoas precisa na maior parte do tempo. Vamos otimizar para aquele cenário. E claro, vamos ter flexibilidade suficiente para fazer alguma outra coisa'" (GEER, 2006, p. 20, tradução nossa).

2.3.6. Ruby - Volt

Volt é um *framework* Ruby que se foca em fazer com que o código escrito seja executado no cliente e no servidor. O documento HTML, escrito utilizando uma linguagem de *templates* própria, atualiza automaticamente dependendo da interação dos usuários com as páginas. O estado de cada página pode ser armazenado na URL (*Uniform Resource Locator*), de forma a manter o estado da aplicação coerente com seu endereço (STOUT, 2016).

O *framework* utiliza programação reativa para, automaticamente, propagar mudanças no documento HTML ou qualquer código que esteja vinculado à um valor no *template*. Quando algo no documento HTML muda, Volt atualiza somente os nodos HTML que precisam ser alterados. Ao invés de sincronizar dados entre o cliente e o servidor através de requisições HTTP, como é o modo de operação da maioria dos *frameworks*, Volt utiliza uma conexão persistente. Quando os dados são atualizados no cliente, os mesmos são atualizados na base de dados ou em quaisquer outras conexões vinculadas. Além disso, Volt tem a opção para a criação de permissões e validações dos dados (STOUT, 2016).

Volt parece fazer parte de uma nova geração dos *frameworks* que utilizam o novo paradigma de programação reativa. Este tipo de *framework* atende à um novo momento da internet, onde um serviço pode estar disponível através de diversos servidores e atende aos

vários aspectos de aplicações modernas. Estes *frameworks* têm por característica responder a eventos de forma assíncrona, logo, podem atender a mais requisições.

2.3.7. Java - Spring Framework

O Spring Framework é uma aplicação de código aberto com o objetivo de minimizar a complexidade associada ao desenvolvimento de aplicações Java. Ao contrário de outros *frameworks* Java, o Spring oferece múltiplas camadas para abranger diversos problemas do desenvolvimento. Atualmente, este é distribuído através da licença Apache 2.0, podendo ser modificado e redistribuído livremente (Kayal, 2008).

De acordo com Kayal (2008), a Inversão de Controle (IOC, em inglês) é o coração de uma aplicação Spring. O IOC simplifica o desenvolvimento das regras de negócio utilizando POJO (*Plain Old Java Object*). O POJO permite a conexão com os variados blocos utilizados para construir o Spring, desde regras de negócios, conexões com bancos de dados, e mapeamento de objetos relacionais (ORM, que simplificam as interações com as bases de dados).

A utilização de IOC pelo Spring também ocasiona a possibilidade de injetar dependências. A Injeção de Dependências (DI - *Dependency Injection*) é um padrão de arquitetura que facilita o desenvolvimento de testes automatizados, boa orientação para a arquitetura da aplicação e promove baixo acoplamento entre os componentes. Este baixo acoplamento entre componentes faz com que o padrão MVC implementado pelo Spring possa ter outros artefatos de *software* conectados à aplicação e, assim, reaproveitar o código (Kayal, 2008).

De fato, o Spring Framework aparenta ser um *framework* muito importante para o desenvolvimento de aplicações em Java. Sua arquitetura em camadas permite que seus componentes sejam de fácil entendimento por programadores, além de lhes permitir a substituição dos mesmos por implementações próprias.

2.3.8. Java - LibGDX

Segundo Nair e Oehlke (2015), LibGDX é um *framework* para o desenvolvimento de jogos. A plataforma se apresenta como solução para desenvolvedores de jogos independentes (conhecidos como ***indies***), cujos custos precisam ser, geralmente, baixos, e o

desenvolvimento, ágil. O *framework* é popularizado em um mercado que cresceu muito nos últimos anos devido ao advento de dispositivos móveis e proliferação de lojas de jogos para computadores.

O LibGDX é multiplataforma. Isto significa que é possível codificar um *software* e executá-lo em mais de um dispositivo, como Android e iOS, computadores pessoais e em navegadores via JavaScript/WebGL. Com o LibGDX, é possível criar cenários dos jogos, adicionar atores, opções, efeitos especiais, transições de tela e adicionar sons. Além disso, a plataforma permite trabalhar com simulações físicas, isto é, simulações de comportamento de corpos em espaços 2D através de uma biblioteca separada (NAIR; OEHLKE, 2015).

Esta plataforma apresenta-se como uma oportunidade para empresas entrantes no mercado ou desenvolvedores independentes. Sua API, explicada por Nair e Oehlke (2015), parece ser simples e expressiva para realizar as ações desejadas. Contudo, sua maior atratividade é o fato de ser multiplataforma, pois esta característica tende a baratear os custos de desenvolvimento.

2.3.9. Python - Flask

De acordo com Grinberg (2014), Flask é um pequeno *framework* para Python. Tão pequeno que pode ser considerado um *micro-framework*. Entretanto, o autor aponta que o fato de ser pequeno, não significa que tenha menos funcionalidades que outros. O Flask provê as funcionalidades principais, enquanto o restante das funcionalidades são providas através de extensões. Grinberg (2014) explica algumas das características do Flask:

• **Rotas e *Views***: o *framework* permite que seu usuário defina rotas e *views* correspondentes. Assim, páginas podem ser renderizadas a partir de rotas específicas. As rotas são criadas utilizando a arquitetura REST.

• ***Templates***: o Flask possui um gerenciador de *templates* nativo, o Jinja2. Estes *templates* são documentos em HTML com alguns espaços reservados para que o *framework* substitua-os durante o processo de renderização de *templates*.

• **Bases de Dados**: o Flask não possui suporte nativo para a conexão e a manipulação em bases de dados. Porém, este objetivo é facilmente atingido utilizando bibliotecas para cada base de dados. Estas bibliotecas, geralmente, oferecem a capacidade de abstrair *queries* e, ao

invés de criá-las manualmente, utilizar classes e métodos que representam os dados. Esta representação é feita através de *Models*.

Aparentemente, de acordo com as características citadas por Grinberg (2014), o Flask é um *framework* leve, utilizado para construção de serviços *web*, ou aplicações simples. Outro *framework* Python popular, que possui mais funcionalidades nativas é o Django, visto na próxima seção.

2.3.10. Python - Django

Django provê uma estrutura de aplicação para que o programador possa focar no código. Foi desenvolvido a partir de 2003, quando programadores da cidade de Lawrence, Kansas, EUA, iniciaram o desenvolvimento de aplicações utilizando Python. Os desenvolvedores iniciaram seu compartilhamento após 2005, e hoje, Django permanece sendo um projeto de código-aberto hospedado no Github (HOLOVATY; KAPLAN-MOSS, 2009).

Entre as características do *framework*, de acordo com Holovaty e Kaplan-Moss (2009), estão: sistema de *template* para renderização dinâmica de páginas, configurações de URL utilizando o padrão REST, interação com o banco de dados através de ORM e sistema MVC para criação de páginas. Uma aplicação construída com Django também possui um sistema administrador para a criação de tabelas do banco de dados com o objetivo de facilitar a criação e o gerenciamento de conteúdo.

Como é possível perceber, muitos dos *frameworks* para o desenvolvimento *web* estão entre os mais populares. Django é um destes *frameworks*, assemelhando-se com Laravel, para PHP e Rails, para Ruby. A explicação para isto pode ser em função da semelhança das próprias aplicações *web*. Muitas são *websites* dinâmicos, que necessitam de acesso à banco de dados e possibilidade de gerenciamento de conteúdo, e têm seu desenvolvimento facilitado por *frameworks* como estes. Além disto, todos eles aceleram e facilitam o desenvolvimento dos projetos aos quais se propõe.

2.4. Resumo do capítulo

Neste capítulo definiu-se e contextualizou-se ecossistemas e o modo como serão utilizados no decorrer deste trabalho. Dessa forma, os conceitos de linguagem de programação e *frameworks* foram definidos.

A escolha das linguagens de programação dá-se pelo serviço de busca de repositórios do Github, considerando as linguagens que possuem o maior número de repositórios estrelados. Este método também é utilizado para a seleção dos *frameworks*. Para a escolha dos mesmos, buscou-se aqueles que estão hospedados no Github e possuem mais estrelas. Deste modo, a seleção dá-se pelas linguagens e *frameworks* mais populares no Github.

Utilizando o método descrito, dez *frameworks* foram escolhidos, dois de cada uma das linguagens mais populares no Github, para serem detectados pelo protótipo. Os *frameworks* selecionados foram: AngulaJs e React (JavaScript); Laravel e Symfony (PHP); Rails e Volt (Ruby); Spring e LibGDX (Java); e Flask e Django (Python).

Com isso, o objetivo específico "**Avaliar as características dos usuários que serão necessárias para o protótipo**" foi atendido, uma vez que as características dos usuários, aqui referidas como ecossistemas (linguagens de programação e *frameworks*), a serem analisadas foram definidas.

3 PROTÓTIPO

Para entender o algoritmo desenvolvido é mais fácil visualizar o funcionamento final do protótipo primeiro. Desta forma, as entradas e saídas descritas na explicação do algoritmo desenvolvido ficam mais compreensíveis. Este capítulo descreve as tecnologias e serviços utilizados, e apresenta imagens do seu funcionamento, relatando como a aplicação foi construída. No capítulo seguinte, explica-se o algoritmo passo a passo.

3.1. Tecnologias utilizadas

É possível dizer que a arquitetura do protótipo possui algumas camadas: o *front-end* que exibe conteúdos ao usuário, o *back-end* que possibilita o processamento das requisições realizadas, o banco de dados para armazenamento, além de serviços de terceiros para a busca de informações. O protótipo foi construído com Ruby on Rails, acessos à API do Github, banco de dados PostgreSQL e integração ao Google Maps[27] para apresentação de um mapa com coordenadas das cidades.

Conforme Flanagan e Matsumoto (2008), Ruby é uma linguagem dinâmica que, mesmo sendo complexa e poderosa, é expressiva a programadores. A linguagem foi criada por Yukihiro Matsumoto, inspirada em Lisp, Smalltalk e Perl, mas "usa uma gramática que é fácil para programadores C e Java aprenderem" (FLANAGAN; MATSUMOTO, p. 2, tradução nossa). Já Rails, como visto no Capítulo 2, foi lançado em 2004 para desenvolvimento de aplicações *web*. É focado principalmente em aplicações que necessitam de banco de dados e utilizam o padrão MVC, além de ter seu foco no princípio de convenção ao invés de configuração.

Outra tecnologia fundamental para a análise dos perfis foi a API do Github que, como já descrito anteriormente, disponibiliza acesso ao vasto volume de dados presentes na rede social. O Github disponibiliza uma biblioteca em Ruby, que integra sua API às aplicações de terceiros que desejam utilizá-la. Assim, é possível acessar informações a respeito de usuários, repositórios e seus conteúdos diretamente.

[27] https://maps.google.com

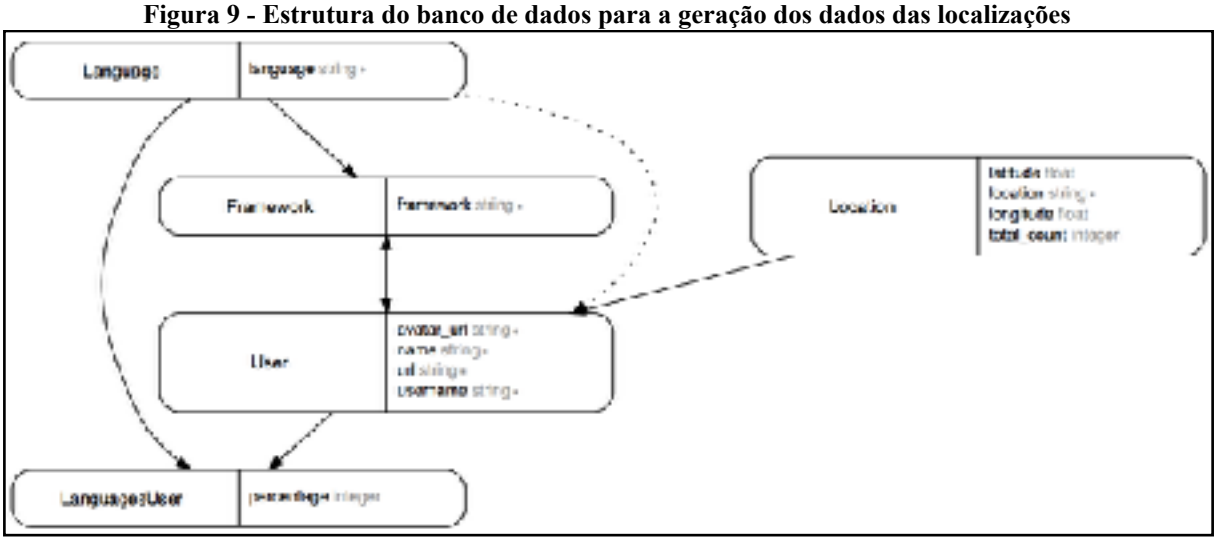
Esta biblioteca, chamada de Octokit[28], possui integração com o *framework* Rails e, consequentemente, agiliza o processo de desenvolvimento. Por exemplo, para realizar uma requisição de um usuário específico à API do Github, precisar-se-ia de bibliotecas para consultar o endereço e enviar parâmetros de URL específicos. Utilizando a Octokit, no entanto, este procedimento é simplificado pelos métodos inclusos na biblioteca, que automaticamente constroem o endereço e parâmetros corretos, e realiza a requisição. A seguinte linha de código retorna o usuário passado como parâmetro:

user = Octokit.user(username)

Esta biblioteca possui métodos para as mais diversas interações com o Github, como alterar informações ou excluir conteúdos. Este protótipo, porém, faz uso apenas da capacidade de leitura de dados e busca na rede social.

O protótipo também se conecta com um banco de dados PostgreSQL. Sua base de dados é utilizada para o salvamento de coordenadas das localidades pesquisadas, junto às informações processadas dos usuários destas localidades. Esta estrutura pode ser vista na Figura 9.

Figura 9 - Estrutura do banco de dados para a geração dos dados das localizações



Fonte: elaborada pelo autor

A tabela de localizações possibilita o salvamento das coordenadas das localidades pesquisadas e usuários encontrados anteriormente. Nesta tabela, as informações armazenadas são: localização (coluna ***location***), coordenadas geográficas (colunas **longitude** e **latitude**) e número de usuários encontrados (coluna ***total_count***). Isto proporciona a exibição de um

[28] https://github.com/octokit/octokit.rb

mapa com os pontos pesquisados e o número de usuários já retornados. Este mapa, por sua vez, é criado utilizando o Google Maps, que possui sua API própria para integração com aplicativos de terceiros, no mesmo modelo do Github.

As tabelas de linguagens, *frameworks* e usuários são utilizadas para armazenar os usuários, suas linguagens e seus *frameworks* quando uma localização é consultada. Assim, não é necessário fazer requisições ao Github quando uma mesma localização é pesquisada mais de uma vez. No entanto, é importante ressaltar que usuários pesquisados em um intervalo de mais de uma semana têm suas informações consultadas no Github novamente, visto que podem ter sofrido alterações neste período.

3.2. Utilização da aplicação

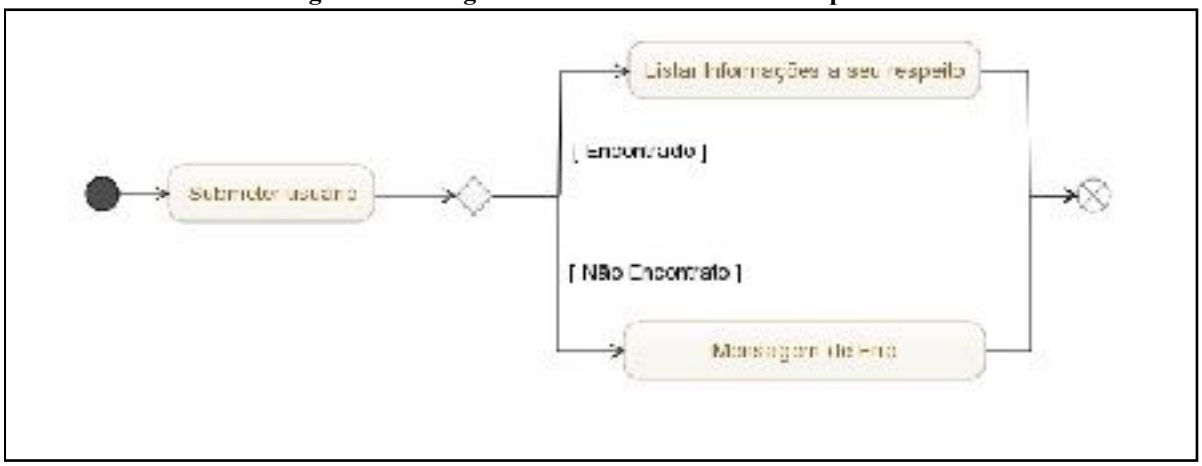
O protótipo tem por objetivo final analisar o perfil de usuários e auxiliar no processo de seleção de candidatos. Para fazê-lo, dois fluxos de execução são disponibilizados: busca por *username* e por localidade. Cada uma destas formas são explicadas e ilustradas nesta seção.

3.2.1. Busca por *username*

O Github tem por característica determinar um nome único para cada usuário. Este nome, chamado de *username*, é uma forma de acessar sua página específica, mencioná-lo em discussões e comentários, ou encontrá-lo via API. Esta última forma é utilizada pelo protótipo para encontrar as informações apropriadas.

Com o intuito de acessar os dados do usuário desejado, o protótipo possui uma página onde um formulário é encontrado com um campo de busca para entrada do *username* desejado. Esta funcionalidade pode ser utilizada para casos onde o usuário em questão já é identificado. Caso o usuário seja encontrado, a sua análise é exposta, caso contrário, uma mensagem de erro é retornada. Tal fluxo de atividades pode ser visualizado na Figura 10. Também, as Figura 11 e 12 apresentam as páginas de busca e de análise retornadas, respectivamente.

Figura 10 - Imagem da análise de um usuário específico

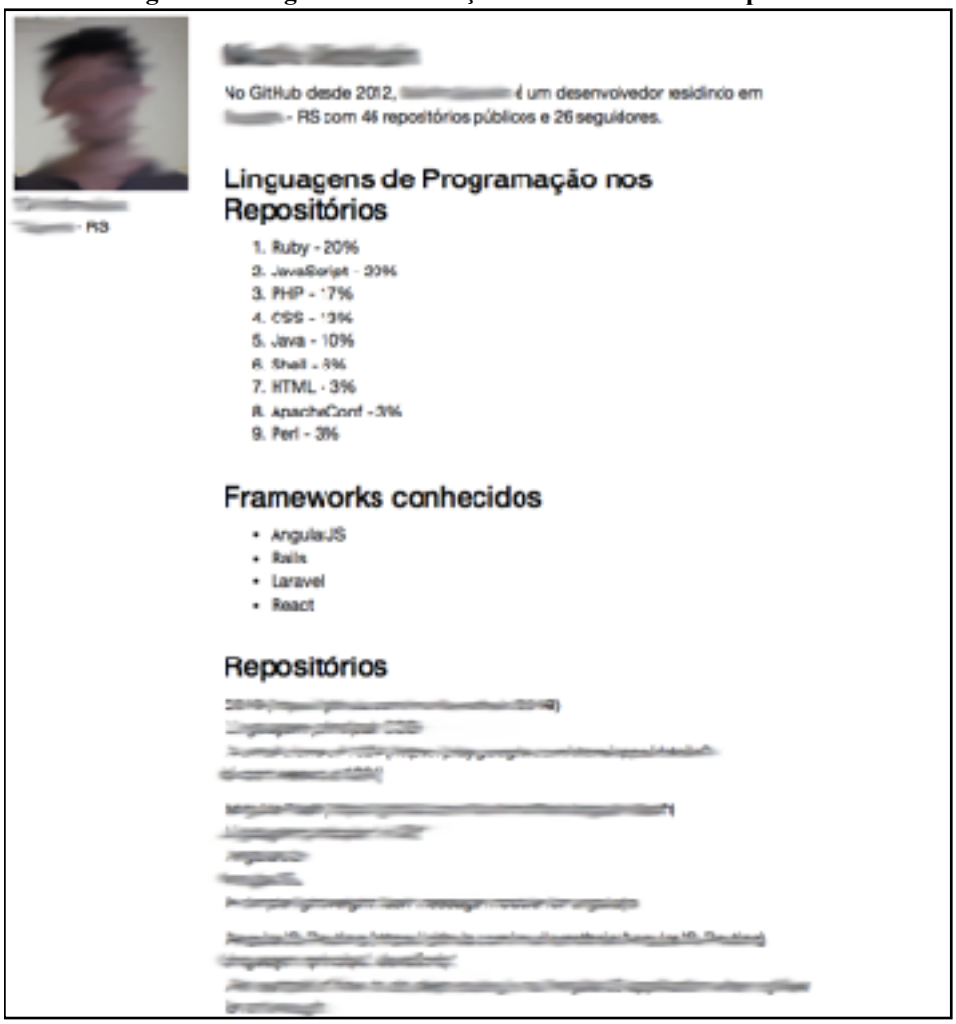


Fonte: elaborada pelo autor

Figura 11 - Página de busca de um usuário pelo seu *username*

Fonte: elaborada pelo autor

Figura 12 - Página de informações sobre um usuário específico



Fonte: elaborada pelo autor

3.2.2. Busca por localidade

O Github também disponibiliza a possibilidade de buscar usuários pela localidade por eles inserida. Porém, é importante ressaltar que a rede social não solicita ao usuário separação de cidade, estado ou país. Como pode ser visto na Figura 13, no campo de localidade é possível inseri-la em qualquer formato. Esta falta de formatação gera problemas na obtenção de usuários por divisões políticas e/ou administrativas, como países, estados e municípios. Para citar um exemplo, programadores de uma mesma cidade podem inserir **Estado, País**, enquanto outros preferem **Município, Estado**. Por esta razão, a geolocalização de desenvolvedores pode não ser precisa e diferir entre diferentes palavras-chave buscadas.

Figura 13 - Formulário de alteração de dados do Github

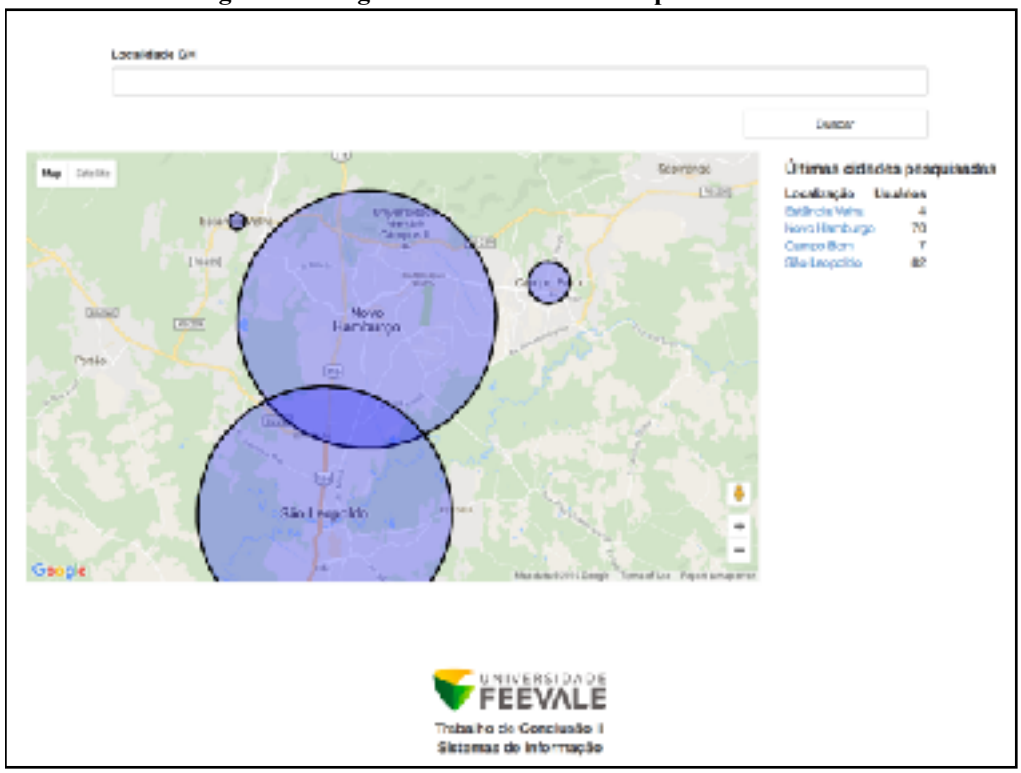
Fonte: Github (2016)

No protótipo, esta busca é feita através de um formulário com apenas um campo, onde os utilizadores inserem a localidade desejada e a eles é retornada uma lista contendo os programadores encontrados no Github e suas principais características, como linguagem de programação de maior fluência, *frameworks* com que trabalha e dados pessoais. Além disso, cada profissional encontrado possui um *link,* que direciona o utilizador ao perfil completo do usuário do Github, ilustrado pela Figura 12. Este fluxo de utilização pode ser visualizado na Figura 14. Nas Figuras 15 e 16 as páginas de busca e listagem de usuários são exibidas, respectivamente. Na Figura 17, o filtro que é mostrado ao usuário após o término do processamento de uma localidade, para que uma seleção baseada em linguagens e *frameworks* desejados possa ser feita, é exibido. As linguagens apresentadas são as encontradas entre as habilidades dos usuários.

Figura 14 - Buscando usuários de uma localização

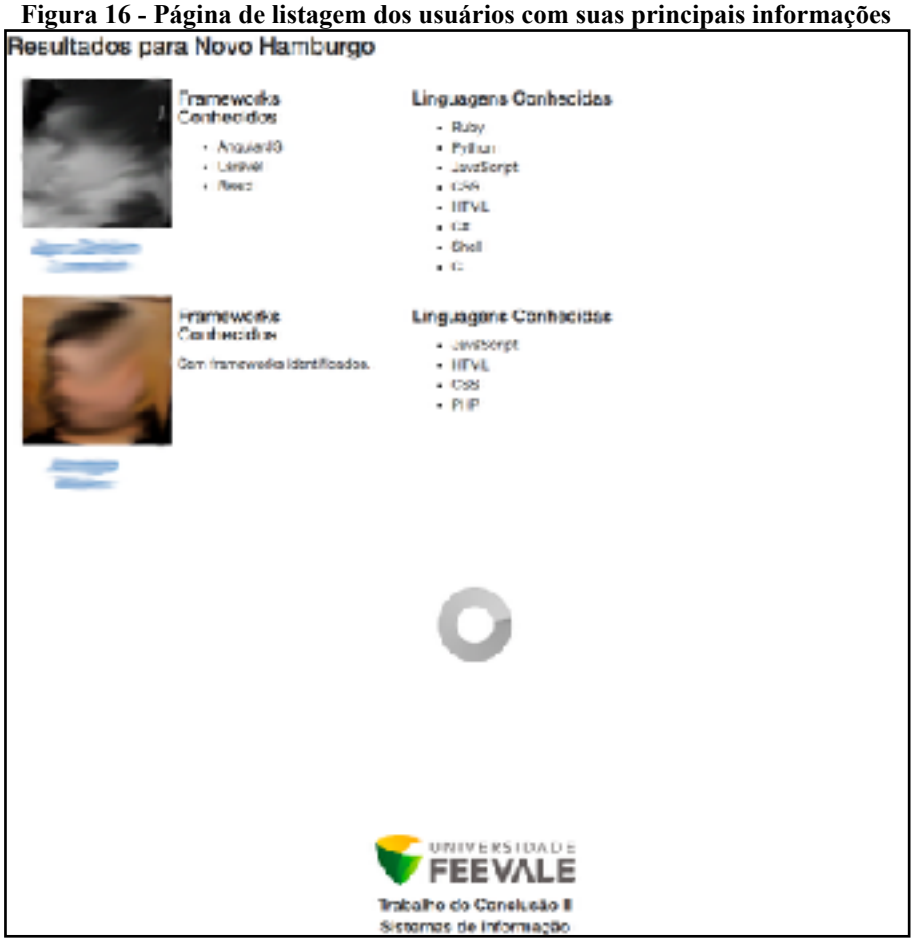
Fonte: elaborada pelo autor

Figura 15 - Página de busca de usuários por localidades



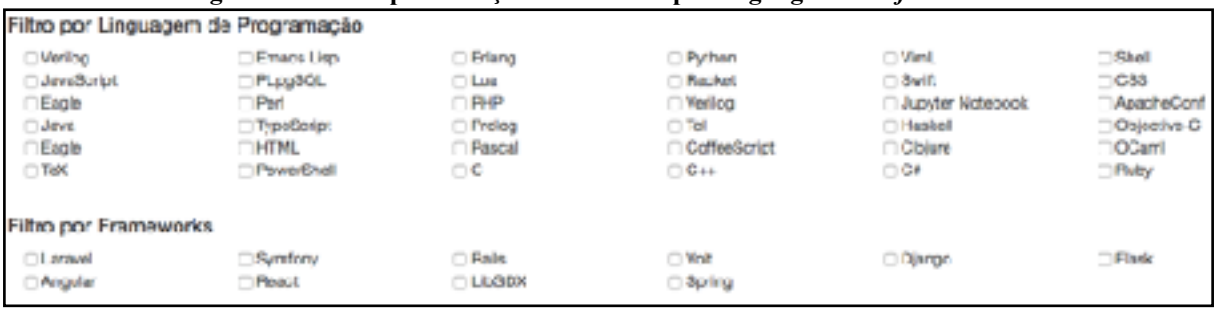
Fonte: elaborada pelo autor

Figura 16 - Página de listagem dos usuários com suas principais informações



Fonte: elaborada pelo autor

Figura 17 - Filtro para seleção de usuários por linguagem e/ou *frameworks*



Fonte: elaborada pelo autor

3.3. Resumo do capítulo

Neste capítulo, o protótipo desenvolvido foi exibido para um melhor entendimento do algoritmo explicado no capítulo seguinte. A escolha das tecnologias, como o *framework*, serviços de geolocalização e acesso ao Github, bem como a linguagem de programação utilizada, deu-se em função da velocidade de desenvolvimento e facilidade do mesmo.

Ruby on Rails, junto à API do Github se mostraram extremamente úteis e fáceis de trabalhar, principalmente em razão da Octokit, ferramenta disponibilizada pelo Github para integração com *softwares* Ruby, que facilita o acesso à sua API. As demais tecnologias listadas também mostraram-se importantes para completar as funcionalidades da aplicação desenvolvida.

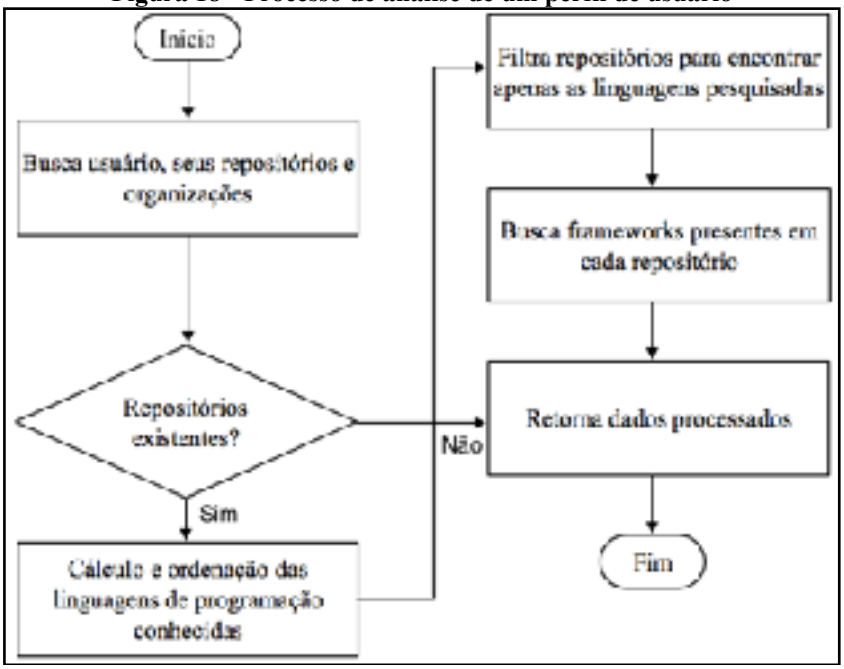
Aqui, também, demonstrou-se a utilização do protótipo através de imagens e diagramas de atividades. É possível, assim, entender o fluxo de eventos para a entrada e saída de dados do programa. Portanto, os objetivos específicos "**Definir as tecnologias para desenvolvimento do protótipo**" e **"Desenvolver um protótipo de sistema baseado em informações coletadas a partir do perfil público do usuário no Github"** foram atendidos.

4 ALGORITMO DESENVOLVIDO

O principal objetivo do protótipo é indicar uma análise de perfil de programadores para processos de seleção. O protótipo utiliza um *username* enviado para efetuar esta análise. A principal característica procurada é o ecossistema no qual os desenvolvedores tenham conhecimento, ou seja, suas linguagens de programação preferenciais e *frameworks* que tenham alguma experiência. A seguir, cada passo desta busca é descrito e discutido.

Devido à busca feita no Github, cujo método foi descrito em capítulos anteriores, definiu-se os *frameworks* a serem pesquisados a partir das linguagens de programação mais populares na plataforma. Esta informação, junto com um resumo a respeito do usuário, é gerada a partir do processamento de dados do perfil de entrada, seguindo o fluxo apontado na Figura 14, no capítulo anterior. Cada uma das etapas apontadas é apresentada na Figura 18 e as mesmas são descritas na sequência.

Figura 18 - Processo de análise de um perfil de usuário



Fonte: elaborada pelo autor

4.1. Processo de análise de perfis de usuários

A entrada inicial do algoritmo é o *username* enviado. Ele é usado fundamentalmente para obter todos os dados usados no decorrer do processo. As primeiras informações a serem buscadas são os dados pessoais do usuário (nome, email e empresa para a qual trabalha), seus repositórios (os quais são usados posteriormente para análise) e as organizações às quais

pertence. Esta busca, como já citado, tornou-se fácil pela utilização da biblioteca Octokit, fornecida pelo Github. Além de tornar a busca do usuário mais fácil, é possível procurar seus dados relacionados utilizando métodos embarcados. O código para buscar o usuário, seus repositórios e as suas organizações, por exemplo, pode ser visto no Quadro 3. As organizações são listadas mais tarde, quando os dados são retornados para visualização, sem afetar o restante da análise do perfil. O usuário e seus repositórios, por outro lado, são utilizados no restante do processo.

Quadro 3 - Busca de um usuário, seus repositórios e suas organizações

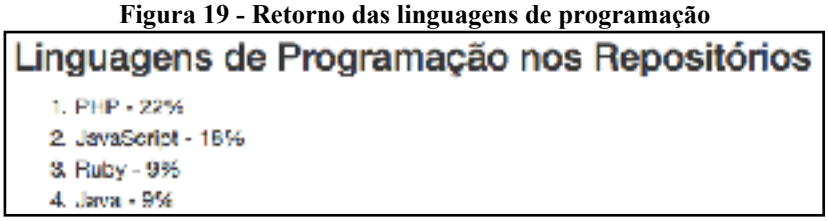
```
1  user = Octokit.user username
2  repositories = user.rels[:repos].get.data
3  organizations = user.rels[:organizations].get.data
```

Fonte: elaborado pelo autor

4.1.1. Cálculo da presença de linguagens de programação nos repositórios

Após a busca dos repositórios públicos do usuário, as linguagens mais utilizadas em cada um são procuradas para deduzir a linguagem de maior fluência do programador. Utilizando-se desta lógica, assume-se que as linguagens mais utilizadas nos repositórios são as de maior familiaridade para o profissional. O resultado é listado por ordem de utilização, como pode ser visto na Figura 19.

Figura 19 - Retorno das linguagens de programação



Fonte: elaborado pelo autor

Este resultado é obtido através de um método de busca, ordenação, e cálculo das linguagens presentes nos repositórios relativas ao total existente. No Quadro 4 é possível ver o método responsável por tal processamento, iniciando-se pela definição do método que recebe os repositórios (linha 1) e finalizando com o retorno das linguagens ordenadas por ordem de importância, na linha 9.

Quadro 4 - Qualificação e ordenação das linguagens de programação

```
1  def qualify_langs(repositories)
2    languages = repositories.map(&:language)
3    languages = languages.select { |x| !x.nil? }
4    total = languages.length
5    languages = languages.group_by(&:to_s)
6    languages = languages.map do |language, count|
7      [language, address_lang_percentage(total, count)]
8    end
9    languages.sort { |x, y| y[1] <=> x[1] }
10 end
```

Fonte: elaborado pelo autor

A partir do recebimento dos repositórios por este método, o programa busca as linguagens presentes nos mesmos, através do método ***map*** (linha 2). Na linha 3, o método ***select*** é usado para selecionar os repositórios cujas linguagens não têm valor nulo. Pode-se citar como exemplo para este caso, repositórios contendo apenas documentações e referências (texto). A variável **total** é criada após este filtro, contento o número total de repositórios, e mais tarde, usada como denominador para relativizar a presença de uma linguagem em comparação ao total de linguagens presentes. Na linha 5, o método ***group_by*** é utilizado para encontrar o total de linguagens agrupadas pela sua linguagem principal. A seguir, na linha 7, o cálculo é feito pelo seu método específico, considerando o total de repositórios e o número de repositórios para cada linguagem de programação. Por fim, o método ***sort*** é usado para retornar os resultados computados em ordem decrescente, como apresentado na Figura 19.

4.1.2. Reconhecimento de *frameworks*

O processamento das informações do usuário-alvo tem por principal objetivo determinar o *framework* com o qual ele atua. Como explicado anteriormente, escolheu-se, baseado em dados de popularidade do Github, 10 *frameworks* que servem de amostragem para este trabalho. Nesta seção, o processo de filtro de cada um deles é descrito.

Quadro 5 - Processo de reconhecimento de um *framework*

```
1  frameworks = {}
2  repositories.each do |repo|
3    repo_content = RepoContent.new(repo)
4    FactoryFilter.filters_from(repo_content).each do |obj|
5      lang = obj.to_s.downcase.to_sym
6      frameworks[lang] = {} unless frameworks[lang]
7      frameworks[lang].merge!(repo[:name].to_sym => obj.frameworks)
8    end
9  end
```

Fonte: elaborado pelo autor

No Quadro 5 é possível visualizar o algoritmo onde os repositórios do usuário são percorridos. No começo do processo (linha 1) a variável responsável por acumular os *frameworks* identificados é iniciada. A seguir, percorrem-se todos os repositórios encontrados e algumas classes externas são utilizadas para auxiliar no processo de reconhecimento. Na linha 3, busca-se, utilizando a classe ***RepoContent***, o conteúdo do repositório, ou nulo, caso o conteúdo do repositório seja vazio.

A classe ***RepoContent*** faz uma busca recursiva nos conteúdos do repositório. Isto é necessário para trabalhar com repositórios que têm suas listas de dependências em diretórios aninhados, e não na raiz do projeto. Este cenário foi identificado em projetos Java, cuja estrutura de arquivos tendem a não seguir convenções como outras linguagens, e projetos diversos que possuíam diretórios dedicados para trabalhar com *front-end* e, por consequência, mantinham seus arquivos JavaScript neles. Como pode ser visto no Quadro 6, o resultado da busca é extraído em um vetor de dados unidimensional com os arquivos e seus endereços. Estes são utilizados posteriormente para leitura.

Quadro 6 - Processo de busca do conteúdo de um repositório

```
1  @paths = []
2  content = begin
3    Octokit.tree(repo[:full_name], "HEAD", {recursive: true})
4  rescue
5    []
6  end
7  @paths = content[:tree].map(&:path) unless content.blank?
```

Fonte: elaborado pelo autor

Depois do processamento de todos os arquivos contidos no repositório, a classe ***FactoryFilter*** retorna uma coleção (linha 4 do Quadro 5), cujo conteúdo corresponde a um

nenhum conteúdo para a linguagem em análise, nada acontece, como pode ser visto nos procedimentos das linhas 5, 6 e 7.

Esta abordagem, utilizando uma classe para a busca de todos os arquivos em um repositório e iterando sobre cada arquivo em busca de um filtro apropriado, foi necessária devido à possibilidade de mais de um *framework* ser usado em um mesmo repositório. Em um primeiro momento, a tentativa feita foi buscar o arquivo de gerenciamento de dependências relativo à linguagem principal do repositório. Por exemplo, em um repositório Ruby, o único arquivo procurado era o **Gemfile**; em um repositório PHP, o filtro procurava por um arquivo **composer.json**. No entanto, alguns repositórios podem conter mais de uma linguagem de programação e, consequentemente, utilizar diversos gerenciadores de pacotes para cada uma delas. Desta forma, a classe **RepoContent** realiza uma busca recursiva (linha 3 do Quadro 6) em todo o conteúdo do repositório e a classe **FactoryFilter** mapeia o filtro relativo aos arquivos encontrados.

Os filtros utilizados são classes independentes que seguem uma mesma interface. Assim, todos os filtros são iniciados com os mesmos parâmetros: repositório e endereço do mesmo. Ainda, todas as classes de filtros acessam uma variável de instância do mesmo tipo (*Hash*), chamada *frameworks*, que permite acesso de leitura pública. Esta abordagem foi utilizada para tornar possível o mapeamento dinâmico entre os objetos de filtros e tornar a entrada e saída de dados comum à todos. Desta forma, ao processar um grande volume de repositórios, para cada repositório cuja linguagem presente tenha uma classe de filtro correspondente, o próprio laço de iteração pode instanciar o objeto apropriado e utilizá-los com a mesma interface.

Como descrito no Capítulo 2, as linguagens e *frameworks* que são objeto de estudo deste trabalho, foram escolhidos através de seu grau de popularidade no Github. Os filtros utilizados para cada linguagem são:

• **PHP**: A comunidade PHP tem utilizado a ferramenta **composer**[29] como gerenciador de dependências. Ambos os *frameworks* utilizados para análise neste trabalho utilizam esta ferramenta para transferir seus arquivos aos projetos. No filtro de PHP, busca-se pelos *frameworks* Symfony e Laravel no arquivo **composer.json**.

[29] https://getcomposer.org/

• **JavaScript:** O gerenciador de pacotes **NPM**[30] vem sendo amplamente utilizado por desenvolvedores JavaScript para distribuírem seus projetos a terceiros. Ele utiliza um arquivo chamado **package.json** para incluir todas as dependências do projeto e baixá-las nas suas versões corretas. Ambos os *frameworks* utilizados para análise neste trabalho utilizam esta ferramenta para transferir seus arquivos aos projetos. No filtro de JavaScript, busca-se os *frameworks* AngularJS e React neste arquivo.

• **Ruby:** As chamadas **gems** são pacotes comumente usados em aplicações Ruby. Ambos os *frameworks* utilizados para análise neste trabalho são distribuídos como **gems**, hospedados no **RubyGems**[31] e utilizam um arquivo denominado **Gemfile** para determinar os pacotes e suas versões. O filtro de Ruby busca neste arquivo por entradas de Rails e Volt para determinar se o repositório contém algum destes *frameworks*.

• **Java:** A linguagem de programação Java possui diversos gerenciadores de pacotes e até mesmo diversos modos de trabalhar com pacotes de terceiros. Em projetos Java, é possível fazer o *download* de um arquivo binário compilado e utilizá-lo, por exemplo. Para os *frameworks* analisados neste trabalho, porém, existem dois gerenciadores recomendados na documentação de ambos os *frameworks*, o **Gradle** recomendado por ambos (LibGDX e Spring Framework) e o **Maven,** recomendado, principalmente, mas não exclusivamente, pelo Spring Framework.

• **Python:** Os *frameworks* analisados neste trabalho, recomendam a utilização do **PIP** (*Pip Installs Python*), um gerenciador de pacotes para Python. Na classe de filtro para repositórios Python, busca-se o arquivo **requirements.txt** e os pacotes em questão, Flask e Django, dentro deste.

Esta abordagem, utilizando arquivos que gerenciam pacotes, não foi a primeira tentativa. Em um primeiro momento, tentou-se comparar arquivos de um repositório com os arquivos iniciais de um *framework*, executando um algoritmo de diferenciação entre ambos e buscando um coeficiente de diferença capaz de categorizar o repositório, ou não. Porém, ao pesquisar sobre as estruturas dos *frameworks* para análise, viu-se que existem alguns problemas com esta abordagem:

• o tempo de processamento seria maior, visto que todos os arquivos seriam comparados com a estrutura pré-estabelecida, principalmente quando o repositório em

[30] https://www.npmjs.com/

[31] https://rubygems.org

questão possuísse um grande número de arquivos;

• seria preciso manter a base de comparação de cada *framework* atualizada, para todas as versões. O Laravel e o Rails, por exemplo, possuem cinco versões, com escritas variadas e, nesta abordagem, seria preciso que cada versão fosse comparada;

• projetos Java são especialmente diferentes, porque os *frameworks* não possuem uma convenção de estrutura de arquivos e diretórios específica, diferentemente das demais linguagens;

• alguns *frameworks* podem disponibilizar alguns módulos para implementações independentes, como é o caso do Laravel e Symfony, onde o primeiro utiliza módulos do segundo. Da mesma forma, usuários podem utilizar módulos de ambos, independentemente, o que não excluiria sua familiarização.

A utilização dos arquivos para o gerenciamento de pacotes tornou esta implementação mais simples, pois *frameworks* de várias versões, ou somente módulos independentes estão discriminados nestes arquivos. Por outro lado, caso os usuários os utilizem de alguma forma não convencional, seja por um gerenciador de pacotes estranho, ou inclusão direta de arquivos, o algoritmo não é capaz de identificá-los. Apesar de possível, estas maneiras não são recomendáveis devido à grande complexidade que adicionar-se-ia à implementação e manutenção de um projeto e, por isto, estes casos são descartados.

A arquitetura utilizando filtros como classes independentes com uma mesma interface, possibilita uma flexibilidade ao trabalhar com múltiplas linguagens e diferentes estruturas de projetos. Como cada linguagem pode possuir um ou mais tipos de gerenciadores de pacotes, a partir da construção de classes independentes como filtros, foi possível realizar este processamento sem influenciar nas demais classes. Por exemplo, a classe **JavaScript**, que pode buscar *frameworks* em arquivos **package.json** e **bower.json** utiliza métodos para fazê-lo, mas retorna apenas os *frameworks* encontrados publicamente, sem necessidade de acoplamento para com o código que a executa. Desta forma, a adição de qualquer filtro passa por adicionar as classes específicas, sem influenciar a arquitetura existente.

4.2. Processo de busca por localização

Porque possui um campo para a inserção da localização, como visto no Capítulo 3, o Github possibilita uma busca pela mesma. Como este campo é de texto livre, não existem

restrições quanto ao formato do seu conteúdo. Ou seja, o usuário insere sua localização baseado em seu critério.

O algoritmo utilizado para a busca dos usuários por localidade é muito semelhante ao algoritmo utilizado para a análise de um perfil, com as seguintes modificações:

- a remoção das organizações, pois essa não é uma informação listada na página de busca;
- os dados encontrados são processados e salvos no banco de dados. Uma vez que estas informações são salvas, o protótipo começa a renderizar os usuários encontrados. Esta abordagem, utilizando um banco de dados, permite que a mesma busca efetuada diversas vezes não precise consultar a API do Github, mas retornar os dados mais rapidamente;
- cálculo de linguagens de programação utilizando apenas a linguagem principal de cada repositório.

Como será discutido no Capítulo 6, as requisições públicas dos usuários foram adicionadas à análise após testes realizados. No entanto, devido à velocidade de execução e limites na quantidade de requisições, a busca por localidade analisa apenas os repositórios próprios de cada usuário encontrado.

O maior problema ao buscar usuários pela sua localidade é, sem dúvida, o limite de requisições dado pelo Github. Segundo o Github Developer (2016), o limite de requisição à sua API RESTful é de 5000 requisições por hora para *download* de recursos específicos e 30 requisições por minuto para buscas. Outro dado importante neste contexto é o número de registros retornados. De acordo com eles, a API de busca retorna, no máximo, 100 registros por página. Para as demais requisições, este número pode variar.

Este limite é fornecido a cada usuário que deseja utilizar a API do Github. A partir de qualquer conta é possível gerar um *token* (sequência de caracteres randômicos) para a identificação na conta. Este *token* é passado como parâmetro a cada requisição enviada, e os limites citados são vinculados a cada um. Por isto, para fins de pesquisa de viabilidade, tentou-se gerar, a partir de duas contas diferentes, dois *tokens* válidos e utilizar ambos com a finalidade de conseguir ampliar o número de requisições possíveis. Apesar de possível, esta implementação foi descartada por se tratar de uma execução não autorizada pela plataforma.

A solução encontrada, então, foi a postergação de requisições até que houvesse limite para tal. Ou seja, ao buscar uma localidade com mais de 3000 usuários, por exemplo, efetua-

se as 30 requisições possíveis no minuto; o processo é parado até que seja possível efetuar mais 30 requisições de busca. Desta forma, este processo demorou cerca de 10 minutos. Esta mesma execução é feita nas buscas dos usuários, onde o procedimento de análise é interrompido quando o mesmo atinge as 5000 requisições possíveis por hora. A análise só prossegue após o *token* ter permissão para mais requisições.

Apesar de não ideal, visto que o processo de análise de grandes regiões pode demorar, esta foi a solução possível sem infringir nenhum termo de usabilidade do Github. Outro problema encontrado durante este processo, foi o número de resultados possível com a API de busca da aplicação. Segundo eles, apenas os primeiros 1000 resultados são disponibilizados a terceiros, ou seja, até a página 10 da busca. Portanto, a busca por localização não pode abranger grandes centros urbanos. Para a região do Vale dos Sinos, por exemplo, este limite é suficiente, pois o número de usuários não ultrapassa o total permitido, mas em tentativas de cidades como Paris e Londres, apenas os primeiros 1000 usuários foram encontrados.

Outro problema para a aplicação da busca por localidade são as palavras-chave inseridas pelos usuários. Se um usuário de Novo Hamburgo, por exemplo, adicionar sua localização como **Brasil**, a busca não o encontrará. A análise sobre esta questão é feita por esta pesquisa e discutida no Capítulo 6.

4.3. Resumo do capítulo

Neste capítulo apresentou-se e discutiu-se o algoritmo desenvolvido. Suas principais partes foram exibidas e comentadas, bem como, o motivo de sua criação. O grande custo computacional do algoritmo dá-se devido à descoberta de possíveis *frameworks* entre os repositórios dos usuário. Outro importante processamento das informações de um usuário sob análise são as linguagens cuja familiaridade é maior. Ao obter estas informações, é possível supor o perfil básico de um programador.

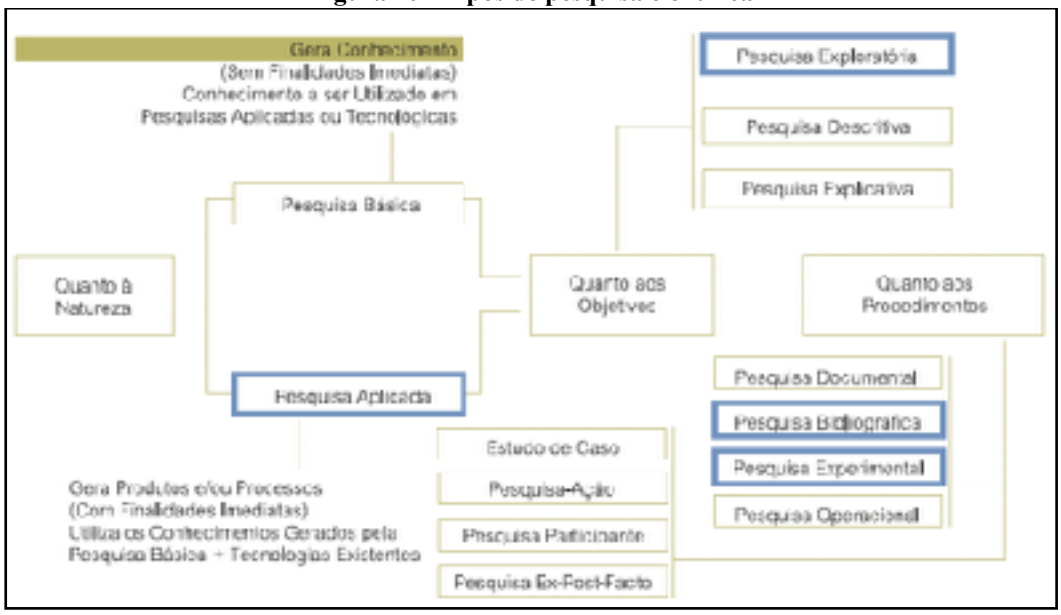
Além disso, foi possível, ainda que superficialmente, buscar programadores pelas suas localidades e analisar o perfil de grandes quantidades de profissionais. Como mostrou-se, no entanto, esta capacidade é deficiente devido aos limites impostos pelo Github e seus termos.

Assim, o objetivo específico **“Escolher o método mais eficaz para a análise de ecossistemas”** foi atendido, uma vez que definiu-se nos sistemas de gerenciamento de dependências e pacotes a fonte para a busca destas informações.

5 METODOLOGIA

Este trabalho tem por objetivo o desenvolvimento de um protótipo de *software* capaz de auxiliar líderes de projetos, administradores, analistas de RH e demais interessados, na tomada de decisão quanto a seleção de candidatos a desenvolvedores. Considerando que o desenvolvimento do protótipo seja para solucionar um problema específico, pode ser caracterizado como pesquisa aplicada (PRODANOV; FREITAS, 2013).

Figura 20 - Tipos de pesquisa científica



Fonte: adaptado de Prodanov e Freitas (2013)

Esta pesquisa pode ser caracterizada como exploratória, uma vez que busca a definição dos conceitos envolvidos em uma temática e utiliza uma amostra para validação do experimento desenvolvido. Os procedimentos adotados são de pesquisa bibliográfica e experimental, sendo que o primeiro se refere aos estudos dos conceitos necessários utilizados como referencial teórico para este trabalho. O segundo visa o desenvolvimento de um experimento e aplicação de uma pesquisa com o público-alvo específico. As características desta pesquisa são identificadas na Figura 20.

A pesquisa experimental permitiu o desenvolvimento do sistema, que deve ser validado. A validação ocorreu em dois momentos: um grupo experimental, denominado de pré-teste, com 13 participantes e um segundo grupo, aqui denominado de pós-teste, composto, também, por 13 participantes (distintos do primeiro grupo), ambos de usuários com conta no Github. Os dois grupos foram compostos, em sua maioria, por alunos da Universidade

Feevale de cursos da área da informática. O formulário de pré-teste gerou modificações no formulário final. É possível, assim, comparar alguns dados que não diferiram entre ambos.

As 26 pessoas foram questionadas através de um questionário *online* contendo perguntas a respeito de suas familiaridades com linguagens de programação, *frameworks* e opinião a respeito do Github. O resultado da primeira análise foi utilizado para modificações no algoritmo para, então, realizar a segunda experimentação. Após a aplicação dos questionários e a análise dos perfis pelo protótipo, questionou-se três entrevistados a respeito de suas respostas.

5.1. Grupo pré-teste

Em um primeiro momento, com a finalidade de entender melhor o público analisado, e as questões a serem abordadas, um experimento de pesquisa foi realizado com uma amostra de 13 pessoas. Todas as pessoas estudam na Universidade Feevale, local onde um formulário de experimentação (Apêndice A) foi aplicado entre os dias 21 e 24 de setembro de 2016. Para responder ao questionário, perguntou-se a todos os participantes a respeito de sua familiaridade com o Github e procurou-se filtrar usuários que possuíssem contribuições públicas nesta rede social.

O objetivo deste filtro é possibilitar um estudo a respeito do algoritmo desenvolvido para entender se as perguntas inseridas responderiam às dúvidas que surgiram após o desenvolvimento do algoritmo. Além disso, este formulário de experimentação forneceu dados para melhoramentos no algoritmo e, posteriormente, realizar a experimentação novamente com um relatório mais completo a respeito do usuário analisado.

Os entrevistados responderam a todas as perguntas. Primeiramente, buscou-se entender características da pessoa, comparando as respostas desta com os dados fornecidos pelo protótipo. Destas respostas, é possível extrair as seguintes informações:

• Em relação aos entrevistados ativos: foi questionado desde quando o respondente é usuário ativo do Github. Dos 13, seis responderam sim e sete responderam não ser ativos.

• Em relação ao tempo: foi questionado desde quando o respondente possui conta no Github. No comparativo entre as respostas do questionário e da consulta automática do *software* no Github, dos respondentes, em apenas um caso não coincidiu a resposta do usuário e do protótipo.

• Quanto à localização, no formulário, todos inseriram a cidade de origem. Porém, analisando no protótipo, chegou-se a conclusão que só quatro entrevistados preencheram este campo no Github, dois indicando Brasil como resposta e não a cidade. Isto pode interferir na busca por localidade do protótipo, pois torna-se impossível buscar todos os usuários de determinada localidade.

• Quanto ao número de repositórios: o formulário perguntava a respeito da quantidade de repositórios com os quais o entrevistado julgava ter trabalhado. A pergunta tinha por objetivo entender se os profissionais, geralmente, trabalham nos seus próprios repositórios públicos. Entre os 13 formulários, nove respostas mostraram-se assertivas em uma comparação entre o relatório emitido pelo protótipo e o número fornecido pelo usuário. Entre as quatro respostas discordantes, duas afirmam ter trabalhado em **mais** repositórios e outras duas afirmavam ter trabalhado em **menos** repositórios que os encontrados pelo protótipo. O fato de um entrevistado acreditar ter trabalhado em mais repositórios que os seus projetos públicos, pode indicar que estes profissionais utilizam repositórios privados, os quais são inacessíveis ao protótipo. Uma vez que na maioria dos casos a resposta foi assertiva, acredita-se que o entrevistado que respondeu menos pode ter se enganado.

• Quanto aos ecossistemas: fez-se, neste momento do questionário, duas perguntas para entender os conhecimentos a respeito dos ecossistemas. Na primeira, perguntou-se o grau de familiaridade (com valores entre 0 e 10) para uma lista de 19 linguagens de programação e, na segunda, quais *frameworks*, entre os contidos no escopo deste trabalho, conhecia e/ou possuía conhecimento prático. Foi possível identificar a presença de *frameworks* em cinco entre os 13 pesquisados. No Quadro 7 é possível ver que apenas dois perfis analisados tiveram os dois *frameworks* assinalados no questionário identificados (Entrevistado 10 e Entrevistado 12). É importante notar que o *framework* Express, inserido pelo Entrevistado 10, não é passível de identificação pelo protótipo. Este pequeno número de *frameworks* reconhecidos, dá-se devido à ausência de repositórios utilizando-os.

Quadro 7 - Relação de respostas dadas comparadas às encontradas pelo protótipo

Entrevistado	Reposta do entrevistado	Resposta encontrada pelo protótipo
Entrevistado 3	Angular/Laravel/Django	Laravel
Entrevistado 6	Angular/Laravel/Spring	AngularJS/Laravel
Entrevistado 10	Angular/Spring/Express	Angular/Spring
Entrevistado 12	Angular/React	Angular/React
Entrevistado 13	Angular/Spring	Angular

Fonte: elaborado pelo autor

O modelo de pergunta utilizado para aferir as capacidades técnicas em relação às linguagens, entretanto, não foi o ideal. Ao tentar analisar os resultados obtidos pelo protótipo a respeito das linguagens, verificou-se que este formato não reflete o retorno apresentado pelo protótipo. Nele, tenta-se ordenar as linguagens pela suposta familiaridade do analisado. Porém, no questionário, pediu-se que o entrevistado selecionasse um valor entre 0 e 10 para o seu grau de familiaridade, tornando difícil uma comparação definitiva para o resultado dado pelo *software*. Portanto, para os questionários a serem aplicados no pós-teste, mudou-se a pergunta, pedindo que as pessoas assinalem a ordem de familiaridade das linguagens, tornando possível a comparação com a resposta dada pelo algoritmo.

• Quanto às vantagens do uso do Github: o questionário também perguntava se o entrevistado acredita que o Github lhe traz alguma vantagem. Todos afirmaram que sim. As justificativas variaram entre colaboração e compartilhamento, acesso rápido à códigos-fonte e gerência dos mesmos, construção de um perfil profissional e obtenção de conhecimento.

Sete entre os 13 entrevistados, acreditam que o Github lhes capacita a colaborar e compartilhar código e projetos com terceiros. As respostas dos entrevistados 2 e 3, respectivamente, mostram um bom exemplo: “compartilhamento de informações, auxiliando a comunidade de desenvolvedores e interessados” e “repositório grátis para compartilhamento de projetos”. Respostas como estas são coerentes com o Capítulo 1, principalmente no que diz respeito às características sociais da plataforma.

Outra resposta dada por cinco entrevistados foi a de que a plataforma é utilizada para gerir projetos e acessar códigos-fonte rapidamente. Segundo o entrevistado 6, para citar um exemplo desta resposta, o Github permite: “Organização de projeto. Time distribuído. Agilidade no desenvolvimento. (...)”. Estas são características importantes, uma vez que são funcionalidades implementadas para tais fins, como, por exemplo, os *pull requests*, explicados no Capítulo 1.

A obtenção de conhecimento é citada por quatro pessoas. Segundo eles, é possível adquirir mais informações a respeito de implementações de outros desenvolvedores de forma prática. “(...) Troca de conhecimento de forma prática” (Entrevistado 6) e “(...) estudar outros *softwares*/ideias” (Entrevistado 1) são exemplos de respostas neste sentido.

Também, é possível extrair que cinco entre os treze entrevistados acreditam que o Github lhes permite criar um perfil profissional e analisar características técnicas dos participantes. Eles citam “[construção de um] portfólio”, e/ou “[construção de um] perfil profissional” (Entrevistados 1, 3 e 8), ou ainda, segundo o Entrevistado 13, “É um forma de reconhecer as características tecnológicas da pessoa”. Como apresentado no Capítulo 1, Dabbish et al. (2012) também mencionam que as pessoas tendem a deduzir habilidades técnicas de profissionais baseando-se em suas contribuições no Github.

5.2. Alterações no formulário

Baseando-se nos resultados obtidos e descritos anteriormente, fez-se alterações em algumas perguntas, bem como adicionou-se outras (Apêndice B). É importante, assim, descrever as alterações efetuadas e seus motivos para um entendimento mais amplo dos resultados desta pesquisa, discutidos no Capítulo 6.

Razão pela qual o usuário se considera ativo: em ambos os formulários, questionava-se se o entrevistado seria, ou não, ativo na plataforma. Isto é importante para questionar se os resultados obtidos pelo protótipo seriam mais acurados para usuários mais ativos, por supor-se que estes teriam mais informações publicadas. No entanto, utilizando o formulário de experimentação, não foi possível estabelecer uma relação entre dados publicados na plataforma e as suas respostas. Dessa forma, adicionou-se uma pergunta questionando, caso o usuário seja ativo, a razão de sê-lo.

Repositórios privados: questionou-se, após analisar os resultados do formulário de pré-teste, se as diferenças entre o número de repositórios os quais o entrevistado afirma ter trabalhado e o número de repositórios encontrados pelo algoritmo, teria acontecido devido ao usuário trabalhar em repositórios privados. Deste modo, adicionou-se a pergunta: “Você trabalha em repositório(s) privado(s)?”, sendo as opções “Sim” ou “Não”.

Finalidade de utilização do Github: ao analisar as respostas dadas pelas pessoas no formulário de teste, questionou-se, para entendimento sobre o perfil de utilização da

plataforma, os motivos que os fazem utilizá-la. Assim, no questionário final, perguntou-se qual a finalidade de uso. Permitiu-se, nesta resposta, assinalar mais de uma resposta.

Uso de linguagens no Github: para fins de validação do funcionamento do algoritmo, na segunda versão da pesquisa, buscou-se entender quais as linguagens de utilização do entrevistado no Github, e não apenas as linguagens que este tem familiaridade. Por isto, do mesmo modo em que se questiona o conhecimento geral, questionou-se as linguagens que o mesmo utiliza na plataforma.

Outra alteração realizada foi a forma como as linguagens de programação eram avaliadas. Ao invés de questionar o grau de familiaridade com valor numérico entre 0 e 10, pediu-se que os respondentes ordenassem as linguagens conforme seu entendimento das mesmas. Isto é importante, pois possibilita a comparação entre a saída do algoritmo e o entendimento do usuário.

5.3. Resumo do capítulo

Neste capítulo, apresentou-se a metodologia utilizada para aplicação do experimento. Com a finalidade de validar o formulário e melhorar os resultados do algoritmo, um grupo de pré-teste foi utilizado. Este grupo, composto por 13 pessoas, expôs falhas na análise de perfis e dificuldade para comparar resultados.

Por isto, este capítulo também explicou as alterações realizadas no formulário e suas razões. Estas referem-se às modificações em perguntas para melhor comparação com os resultados obtidos e adição de outras, uma vez que o grupo pré-teste evidenciou falta de contexto para responder algumas questões. Além disso, foi explicado que entrevistas ocorreram após a análise dos perfis para elucidação das respostas.

6 RESULTADOS E DISCUSSÃO

Este capítulo discute os resultados obtidos comparando os dados gerados pelo protótipo e os apresentados pelos entrevistados. O objetivo aqui é descobrir se as informações públicas cadastradas pelos usuários registrados no Github são suficientes para aferir as características técnicas dos profissionais. Para encontrar o melhor algoritmo possível, testou-se o perfil dos entrevistados da seguinte forma: 1) gerou-se a análise de cada entrevistado utilizando o mesmo algoritmo utilizado para o grupo pré-teste; 2) alterou-se o algoritmo para buscar contribuições públicas em outros repositórios e a forma como as linguagens são ordenadas; e 3) gerou-se, para cada respondente, seus respectivos relatórios.

Primeiramente, do mesmo modo que o formulário de experimentação, buscou-se entender as características da pessoa, comparando as respostas desta com os dados fornecidos pelo protótipo. Destas respostas, é possível extrair as informações a seguir.

• **Relativas aos entrevistados ativos:** dos 13, dez responderam ser ativos e três afirmaram não ser ativos. Quando questionados sobre o motivo que os fazem ativos, cinco disseram que armazenam seus projetos na plataforma, seja para compartilhamento, consultas futuras ou para trabalhar em projetos de código aberto. Duas pessoas afirmaram ser ativas por utilizar a plataforma para aprendizado e acompanhar fóruns (uma pessoa disse que raramente faz contribuições, mas é ativa por utilizar o sistema para este fim). Um entrevistado disse ser ativo porque responde às questões postadas em seus projetos o mais rápido possível e outro respondeu ser ativo por acessar a rede social diariamente. Houve, ainda, um usuário que se diz ativo por ter "prazer em aprender com [*software* de código aberto]" (Entrevistado 13).

• **Em relação ao tempo:** no comparativo entre as respostas do questionário sobre o ano de inscrição na plataforma e o da consulta automática do *software* no Github, dos respondentes, não houve nenhum caso que não coincidisse.

• **Quanto à localização:** nas entrevistas todos inseriram a cidade de origem, algumas vezes incluindo o estado ou país. Porém, analisando no protótipo, chegou-se a conclusão que nove entrevistados preencheram este campo no Github, dois indicando **Brazil** como resposta, um inseriu **RS/Brazil** e não indicou a cidade, e um inseriu uma cidade fictícia. Os demais não inseriram sua localidade no Github. Ao analisar estes dados, sob a perspectiva de busca por localização, percebe-se que, buscando por cidade, seria possível encontrar cinco, entre os

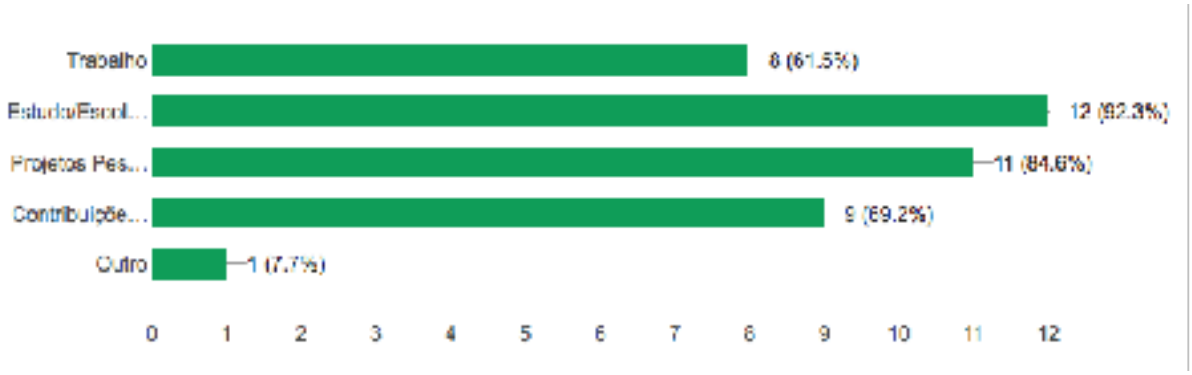
treze respondentes. Isto indica que, em função do mal preenchimento, a busca por cidade não retornará todos os usuários.

Para entender a motivação da inserção dos dados, entrevistou-se três respondentes, interpelando-os a respeito das divergências encontradas. Entre as perguntas realizadas, questionou-se ao Entrevistado 10 o porquê de ele não ter inserido sua cidade. Segundo ele, a não inserção da informação deve-se à falta de vontade, apenas. O Entrevistado 3, cuja resposta foi seu estado e país, disse que o motivo é "não inibir ofertas de emprego". Estas respostas demonstram que o campo "Localidade", provavelmente por não referir-se a nenhuma informação específica, como cidade ou país, permite ao usuário preenchê-lo, ou não, conforme seu critério e interesse.

• **Quanto à utilização de repositórios privados:** sete entrevistados afirmaram trabalhar em repositórios privados e seis disseram não trabalhar.

• **Quanto à finalidade de uso do Github:** foram fornecidas cinco opções de múltipla escolha: trabalho, estudo/escola/universidade, projetos pessoais, contribuições em projetos *open source* e outro. Como pode ser visto no Gráfico 3, 12 entrevistados utilizam o Github para fins educativos, 11 para projetos pessoais, 9 para contribuições em projetos de código livre, 6 para trabalho e apenas 1 tem outra finalidade. Entre os sete entrevistados que trabalham em repositórios privados, seis têm no Github o trabalho como uma das finalidades, o que mostra que, geralmente, usuários não utilizam repositórios privados para fins pessoais ou estudo. Apenas um destes disse não utilizar a plataforma para fins de trabalho, mas tem repositórios privados. Além disto, o entrevistado cuja resposta foi "Outro" explicou: "explorar novas ferramentas, ler código de bibliotecas, acessar *issues* de projetos que utilizo" (Entrevistado 3).

Gráfico 3 - Finalidade de uso do Github



Fonte: elaborado pelo autor

• **Quanto às vantagens do uso do Github:** todos os entrevistados afirmaram que o Github lhes traz alguma vantagem. Entre as justificativas citadas estão: visibilidade por parte do mercado, armazenamento e compartilhamento de projetos, contribuições em bibliotecas e outros projetos, e obtenção de conhecimento.

Quatro entrevistados citaram que o Github lhes traz visibilidade no mercado. “ele é um portfólio/currículo muito bom, pois mostra minhas habilidades” (Entrevistado 3) e “além dos benefícios óbvios da plataforma, creio que hoje o Github vale mais do que um bom currículo” (Entrevistado 4). Essas são duas respostas que mostram a importância da plataforma para este fim. Estas falas concordam com Dabbish et al. (2012), como demonstrado no Capítulo 1, que afirmam ter no Github uma fonte de análise de perfil de desenvolvedores.

O compartilhamento e armazenamento de repositórios também é citado por quatro respondentes como fator importante. Segundo o Entrevistado 3, ele pode “não apenas utilizar, mas também disponibilizar ferramentas/[bibliotecas]/*frameworks* através dele”, e o Entrevistado 1, que diz utilizar a plataforma para “armazenamento seguro”.

As contribuições em projetos de terceiros aparecem na segunda fase de entrevistas como um fator vantajoso. O Entrevistado 7, por exemplo, diz utilizar para ter “contato com comunidade”. O Entrevistado 13 diz que “[quanto a] *frameworks,* facilita a interação com o fluxo de desenvolvimento”. Este tipo de citação diferiu das respostas dadas pelos primeiros entrevistados, acredita-se que é devido aos últimos serem mais ativos que os primeiros.

Da mesma forma que os primeiros respondentes, a troca de conhecimento parece ser a maior razão da utilização do Github. Cinco entre os treze citaram-na como fator importante. Segundo o Entrevistado 7, ele pode ter contato com “fóruns de desenvolvimento”. O Entrevistado 10 diz que a rede social é importante para “compartilhamento de conhecimento e ideias” e o Entrevistado 11 parece concordar, afirmando que é importante para a “troca de conhecimento”.

Quando comparam-se as motivações entre ambos os grupos, isto é, respondentes do pré e pós-teste, percebe-se que há semelhanças entre as motivações. A maior diferença dá-se nos tipos de interação e contribuições em mais projetos ou seus próprios, provavelmente porque o segundo grupo é composto de usuários mais ativos (10 usuários disseram ser ativos

no segundo grupo e 6 no primeiro) e trabalharem, em média, em mais repositórios que o primeiro grupo.

6.1. Alterações feitas no protótipo

Ao verificar os resultados obtidos pelo protótipo em comparação às respostas dadas pelos entrevistados, percebeu-se que seria possível melhorá-lo realizando algumas alterações no algoritmo. Por isso, para a análise dos ecossistemas dos respondentes da segunda versão do questionário, gerou-se duas versões: uma para o mesmo algoritmo executado com o grupo pré-teste e outra com modificações, como adição das contribuições públicas geradas na plataforma e demais linguagens presentes nos repositórios, como explica-se a seguir.

A primeira modificação realizada foi na busca por linguagens de programação presentes. Como explicado no Capítulo 4, a linguagem de maior presença no repositório é utilizada para o processamento da familiaridade do programador. Supõe-se que a linguagem com maior presença de trabalho será aquela com a qual o profissional tem maior fluência. No entanto, na primeira versão do algoritmo, buscou-se apenas pela linguagem de maior presença em cada repositório, não considerando que muitos repositórios possuem diversas linguagens. Assim, a primeira modificação efetuada foi na forma como estas são buscadas e processadas.

Nesta segunda versão, em cada repositório do usuário, consulta-se a API do Github em busca da quantidade de *bytes* de cada linguagem presente. É possível, assim, calcular a quantidade de cada linguagem relativa às linguagens presentes em todos os repositórios, ao invés das linguagens principais. Após este procedimento, o total de cada uma é adicionado a um somatório para, posteriormente, possibilitar a definição de sua porcentagem relativa ao total de linguagens presentes em todos os repositórios. A diferença entre esta versão e o método apresentado no Capítulo 4, dá-se em função da requisição de todas as linguagens presentes em um mesmo repositório, como pode ser visto no Quadro 8. O restante do procedimento é equivalente ao apresentado no Capítulo 4.

Quadro 8 - Requisição dos *bytes* das linguagens presentes em cada repositório

```
1  languages = repositories.inject([]) do |languages, repo|
2    languages << repo.rels[:languages].get.data
3  end
```

Fonte: elaborado pelo autor

Além do cálculo das linguagens de programação analisadas, outra alteração foi a quantidade de repositórios buscados. Para possibilitar uma análise mais ampla, adicionou-se ao algoritmo os repositórios que receberam *commits* através da ferramenta de *pull requests*, como explicado no Capítulo 1. Estas contribuições podem ocorrer em qualquer repositório hospedado no Github que esteja aceitando contribuições.

Para fazê-lo, o algoritmo incluiu mais uma requisição à API do Github, também utilizando a biblioteca Octokit, como explicado no Capítulo 2, executando a linha 1 do Quadro 9. O restante do procedimento é semelhante ao algoritmo apresentado no Capítulo 4, onde itera-se sobre cada repositório e aplica-se o filtro necessário. Desta forma, o universo de repositórios abrangidos é maior, fornecendo mais dados para análise.

Quadro 9 - Requisição dos *pull requests* aceitos

```
1 prs = Octokit.search_issues("type:pr is:merged author:#{username}")
```

Fonte: elaborado pelo autor

As alterações efetuadas proveram mais informações a respeito dos usuários, como número de repositórios que trabalharam, maior eficácia no número de linguagens com as quais trabalharam, e o número de *frameworks* encontrados. Na seção seguinte, comparam-se os resultados de ambos algoritmos.

6.2. Resultados obtidos comparando-se as duas versões do algoritmo

As alterações realizadas no algoritmo afetam alguns pontos da análise, como número de repositórios, linguagens de programação e *frameworks* identificados. Nesta seção, discutem-se os resultados obtidos, comparando-os às duas versões do algoritmo.

Quanto à quantidade de repositórios: entre os 13 formulários, seis respostas mostraram-se assertivas em uma comparação entre o relatório emitido pelo protótipo com a primeira versão do algoritmo. Comparando-se a segunda versão, percebe-se que este número desce para cinco. No que diz respeito à segunda versão, o número de repositórios trabalhados pode, somente crescer, pois inclui contribuições públicas efetuadas. Estes dados mostram que o protótipo acertou em menos de 50% das respostas. Todas as respostas são listadas no quadro 10.

Quadro 10 - Comparação entre as duas versões do algoritmo e os repositórios identificados

Entrevistado	Resposta dada	Primeira versão	Segunda versão
Entrevistado 1	Entre 1 e 5	1	1
Entrevistado 2	Entre 1 e 5	46	46
Entrevistado 3	Mais de 20	28	41
Entrevistado 4	Entre 6 e 10	18	19
Entrevistado 5	Entre 1 e 5	0	0
Entrevistado 6	Entre 16 e 20	14	14
Entrevistado 7	Entre 6 e 10	8	11
Entrevistado 8	Mais de 20	29	44
Entrevistado 9	Entre 1 e 5	4	4
Entrevistado 10	Entre 6 e 10	22	23
Entrevistado 11	Entre 11 e 15	45	51
Entrevistado 12	Entre 11 e 15	4	29
Entrevistado 13	Mais de 20	64	83

Fonte: elaborado pelo autor

Entre as respostas discordantes, algumas destacam-se. O Entrevistado 5, ao ser perguntado no momento da aplicação da pesquisa se havia trabalhado com o Github, respondeu afirmativamente. Destacou, porém, somente ter contribuído em repositórios de terceiros, não possuindo nenhum repositório em sua própria conta. Como este é um cenário importante para a verificação do protótipo, prosseguiu-se com as perguntas a esta pessoa. No entanto, após as modificações no algoritmo, para incluir contribuições públicas à terceiros, e execução da análise no perfil do Entrevistado 5, percebeu-se que este não havia contribuído utilizando a ferramenta disponibilizada pelo Github, mas enviado código diretamente através de permissão de escrita direta. Desta forma, não foi possível rastrear o código enviado e, consequentemente, não exibiu-se nenhum resultado para esta pessoa.

Outra resposta destacada foi dada pelo Entrevistado 2. Ele afirma ter trabalhado em, no máximo, cinco repositórios, quando foram encontrados 46 pelo protótipo, em ambas as versões do algoritmo. Uma entrevista foi realizada posteriormente com esta pessoa e, quando questionado a respeito desta diferença, afirmou que referiu-se, em sua resposta, a repositórios ativos, desconsiderando outros com os quais já tivesse trabalhado: “um a cinco [repositórios] que eu trabalho no momento”, disse. Esta interpretação é corroborada pelo Entrevistado 10.

Ele também afirmou ter trabalhado em até 10 repositórios, quando foram encontrados, nas duas versões dos algoritmos, 22 e 23, respectivamente. Quando entrevistado, posteriormente, disse que considerou apenas os últimos que trabalhou.

Estas informações divergentes e as explicações dos entrevistados são importantes para demonstrar a grande subjetividade presente nas respostas dos entrevistados. Ainda que a pergunta seja explícita, ela é muito abrangente. Portanto, os profissionais podem diferir com relação a sua interpretação. Por isto, é possível concluir que os dados referentes ao número de repositórios encontrados pelo protótipo não são incorretos. As demais divergências podem dever-se ao mesmo fator, engano, ou trabalho em repositórios privados, como, por exemplo, o Entrevistado 6, que afirmou trabalhar em repositórios privados e possui menos repositórios públicos encontrados que sua resposta.

Quanto à identificação das linguagens de programação: diferentemente do experimento realizado com o grupo pré-teste, no questionário final não se pediu para os entrevistados avaliarem seus conhecimentos sobre as linguagens com uma nota (entre 0 e 10). Como viu-se, não foi possível avaliar corretamente os resultados do protótipo a partir desta métrica. Então, a pergunta foi modificada para que o usuário ordenasse as linguagens conforme sua própria familiaridade, uma vez que esta é a mesma exibição trazida pelo protótipo. Devido às diferentes versões do algoritmo, existem dois resultados para esta análise: um dado pela primeira versão do algoritmo (Apêndice C), e outro dado pela segunda versão (Apêndice D).

Em um primeiro momento, ao ler os dados, percebe-se que a ordem assinalada nos formulários e a ordem exibida pelo protótipo são diferentes para todos os respondentes. Porém, ao conversar com os entrevistados, ficou claro que suas avaliações a respeito do conhecimento de linguagens e, principalmente, a ordem que responderam é, de fato, subjetiva e difícil de determinar a partir de métricas discretas, como quantidade de *bytes* das linguagens em cada repositório. Isto é, não é possível determinar, de maneira uniforme a todos, todas as características que são descritas pelos próprios analisados de formas diferentes, pois eles podem seguir critérios próprios para determinar se sabem e o quanto sabem sobre certa linguagem e/ou *framework*.

O Entrevistado 10, por exemplo, ao ser exposto às diferentes respostas entre sua primeira opinião, disse que, “pensando melhor eu até mudaria a ordem do Ruby com o

Objetive-C". Em sua resposta, ele disse que a ordem seria 1) JavaScript; 2) C#; 3) Java; 4) Ruby; 5) Objective C; e 6) Perl. Ao refletir novamente, então, teria mudado as posições de Ruby para 5 e Objective-C para 4. Outra diferença apontada posteriormente com o Entrevistado 10 foi com relação à Java. Segundo a ordem informada, Java seria a terceira linguagem com a qual teria maior familiaridade e, de acordo com o protótipo, Java seria a segunda. Conforme o Entrevistado 10, esta diferença dá-se em função de Java ser utilizado nas disciplinas da universidade, sendo que os repositórios que a contém são mais antigos e estão inativos. Sua preferência, segundo ele, seria por JavaScript ou C#.

O Entrevistado 3 assinalou a seguinte ordem: 1) Ruby; 2) Java; 3) JavaScript; 4) Clojure; 5) C#; 6) CoffeeScript; 7) Shell/ShellScript; 8) Python; 9) C; 10) Haskell; 11) Lua; 12) Matlab; 13) Perl; e 14) Objective C. A primeira versão do algoritmo encontrou as seguintes linguagens: 1) Ruby; 2) Clojure; 3) JavaScript; 4) Java; 5) TypeScript; 6) CoffeeScript; e 7) Perl. A segunda versão encontrou seus repositórios próprios e repositórios de terceiros que fez contribuições. A ordem encontrada de seus repositórios foi a seguinte: 1) Ruby; 2) JavaScript; e 3) TypeScript. A ordem encontrada para repositórios de terceiros que contribuiu foi: 1) Ruby; 2) Clojure; 3) JavaScript; e 4) CoffeeScript. Estes resultados são um exemplo de que o algoritmo não é capaz de ordenar as linguagens com as quais os respondentes afirmam ter familiaridade, mas reflete as linguagens que o entrevistado já trabalhou. Isto é, apesar de não ser possível estabelecer a mesma ordem, é possível verificar que as linguagens trazidas são, ou as mesmas, ou muito semelhantes.

Quando perguntado, o Entrevistado 3 afirmou que não lembrava-se de nenhum repositório em TypeScript. Porém, ao refletir, lembrou-se que existe alguma biblioteca em que havia corrigido um *bug*, e que esta utilizava a linguagem. Esta, de fato, foi a explicação que deu para as linguagens que aparecem em suas contribuições públicas, na segunda versão do algoritmo. Segundo ele, sempre que utiliza algum pacote de terceiros e encontra uma funcionalidade defeituosa, abre um *pull request* com a correção. Aliás, de acordo com ele, este é um dos motivos de não ser possível identificar contribuições em Java, já que a maioria das bibliotecas que utiliza não estão hospedadas no Github. A outra razão que citou foi o fato de os repositórios que trabalha com esta linguagem serem privados.

O Entrevistado 2 afirmou que a ordem de familiaridade com as linguagens que conhece é: 1) JavaScript; 2) Ruby; 3) PHP; e 4) Java. Quando confrontado com as respostas

dadas pelo algoritmo em sua segunda versão, que é 1) PHP; 2) JavaScript; 3) Ruby; e 4) Shell, disse que a alta presença de linguagem PHP deve-se ao fato de ter estudado e trabalhado com PHP há algum tempo e ter publicado este código no Github. Também disse que nesta época era mais ativo na plataforma, se comparado ao momento atual. Isto se mostra como outro fator relevante à análise: se o usuário utilizou a aplicação durante um tempo para determinada linguagem, consequentemente, esta será a linguagem mais relevante como resultado. Ainda, a linguagem Java não aparece em nenhum repositório porque, conforme disse, trabalhava apenas academicamente.

É importante, neste ponto, ressaltar os indícios de subjetividade com que cada indivíduo trata seus próprios conhecimentos em linguagens ou *frameworks*. Esta subjetividade parece ser ainda mais determinante para a ordenação dos conhecimentos de cada um. Por exemplo, o Entrevistado 2 esclareceu posteriormente que considerou apenas as linguagens com que atua no momento. O Entrevistado 10, após reflexão, acredita que a ordem seria diferente da que informou em um primeiro momento. Isto mostra que até mesmo para as pessoas-alvo do estudo, a opinião a respeito de fluência em linguagens é difícil de determinar por ser dependente de idiossincrasias.

Outro problema que surgiu a partir da segunda versão do algoritmo foi a ordenação das linguagens de repositórios próprios e linguagens de repositórios com contribuições. A ordem dada em um primeiro momento é independente entre elas. A hipótese pensada foi a de concatenar as listas, formando apenas uma hierarquia de linguagens. No entanto, após entrevistas e análise das informações supracitadas, percebeu-se que este esforço seria inútil, uma vez que os próprios entrevistados teriam opinões diferentes entre si e, dependendo do ponto de vista, opinões diferentes para seu próprio perfil. A solução utilizada, então, foi a de comparar as respostas como listas não-ordenadas, verificando a presença das linguagens, apenas.

Utilização de linguagens no Github: para entender melhor este cenário, perguntou-se quais linguagens os entrevistados usam na plataforma. A intenção era entender se os dados trazidos pelo protótipo refletem, de fato, esta realidade. No Quadro 11 é possível ver o resultado obtido pela segunda versão do algoritmo e as respostas dadas pelos entrevistados.

Quadro 11 - Comparação entre as linguagens que os usuários dizem utilizar no Github e as linguagens encontradas pelo protótipo

Entrevistado	Linguagens identificadas pelo protótipo	Linguagens que o entrevistado usa no Github
Entrevistado 1	Java; JavaScript	PHP; Java; JavaScript
Entrevistado 2	PHP; JavaScript; Ruby; Shell	JavaScript; Ruby
Entrevistado 3	Ruby; JavaScript; TypeScript; Clojure; CoffeeScript	JavaScript; Ruby; Java; C#; Clojure
Entrevistado 4	Jupyter Notebook; PHP; JavaScript; Python; Java	PHP; JavaScript; Ruby; CoffeeScript; Python; Java; Shell/ShellScript; C; C#; Haskell; Clojure; Go; Lua; Matlab; Perl; R; Scala; Objective C; VimL
Entrevistado 5	-	Java
Entrevistado 6	JavaScript; Java	Java; JavaScript
Entrevistado 7	Java; Python; JavaScript; SQLPL	Java; JavaScript; Python; Shell/ShellScript
Entrevistado 8	C; JavaScript; C++; Java; Python; Ruby; CoffeeScript; Groovy; Objective C	JavaScript; Python; Java; C#; Shell/ShellScript
Entrevistado 9	PHP; Java	PHP; Java
Entrevistado 10	JavaScript; Java; C#; Erlang	JavaScript; C#; Java
Entrevistado 11	JavaScript; Ruby; C#; PHP; Elixir	JavaScript; Ruby
Entrevistado 12	JavaScript; Ruby; Shell/ShellScript; Java; C; Cucumber	Ruby; JavaScript; CoffeeScript
Entrevistado 13	Crystal; Matlab; Ruby - JavaScript; Shell/ShellScript; C; VimL	Ruby; CoffeeScript; JavaScript; Go

Fonte: elaborado pelo autor

Como é possível ver, a maior parte das linguagens identificadas pelo protótipo, constam nas linguagens que os entrevistados trabalham no Github. Mesmo assim, o fator de subjetividade aparece na análise destes dados. O Entrevistado 2 selecionou as linguagens que trabalha atualmente (“Ruby e JavaScript”, segundo disse) e desconsiderou os repositórios inativos. O Entrevistado 3, por sua vez, afirmou que Java e C# são linguagens que usa no trabalho e, por isso, não aparecem publicamente. O algoritmo, como já mencionado, busca informações em todos os repositórios, o que explica a diferença para estes dois entrevistados.

Outra resposta a se destacar é a do Entrevistado 4. Ele afirmou utilizar as 19 linguagens listadas quando o algoritmo encontrou apenas 5. Porém, uma das linguagens encontradas no Github foi a Jupyter Notebook, que segundo sua documentação[32], é uma aplicação *web* que permite a portabilidade para diversas linguagens. Isto pode sugerir que o entrevistado baseou-se nesta característica para assinalar linguagens que tem familiaridade.

[32] Pode ser vista em: http://jupyter.org.

Quanto à identificação de *frameworks*: outro importante ponto na análise de perfis proposta é a identificação dos *frameworks* os quais o indivíduo conhece. No Quadro 12 pode-se ver as respostas dadas e as respostas achadas por ambas as versões do algoritmo. Verifica-se, neste quadro, que houve melhora na identificação dos *frameworks* quando os algoritmos são comparados. Para os entrevistados 8, 11 e 12, foi possível identificar mais *frameworks* na segunda versão, quando inclui-se no escopo da busca os repositórios de terceiros os quais o usuário contribuiu.

Quadro 12 - Comparação entre as duas versões do algoritmo para os *frameworks*

Entrevistado	Resposta dada	Primeira versão	Segunda versão
Entrevistado 1	Code Igniter	-	-
Entrevistado 2	AngularJS; React; Rails	AngularJS; Rails; Laravel; React	Angular; Rails; Laravel; React
Entrevistado 3	AngularJS; Rails; Spring Framework	Rails; AngularJS	Rails; AngularJS
Entrevistado 4	AngularJS; React; Laravel, Rails; Phalcon	-	-
Entrevistado 5	-	-	-
Entrevistado 6	AngularJS	-	-
Entrevistado 7	AngularJS; Spring Framework; SpringBoot	Spring Framework	Spring Framework
Entrevistado 8	AngularJS; React; Rails; Spring Framework	React; AngularJS	AngularJS; React; Rails
Entrevistado 9	Laravel	Laravel; Symfony	Laravel; Symfony
Entrevistado 10	AngularJS; React	AngularJS	AngularJS
Entrevistado 11	React; Rails	Rails; React	Rails; React; AngularJS
Entrevistado 12	AngularJS; React; Rails; Spring Framework	Rails	Rails; AngularJS; React
Entrevistado 13	Rails; Volt; Rust/Crystal/Elixir	Rails	Rails

Fonte: elaborado pelo autor

No entanto, como é apresentado neste quadro, apenas os entrevistados 2 e 9 tiveram todos os *frameworks* que citaram identificados. Os demais não possuíam repositórios contendo *frameworks*, nem haviam contribuído publicamente em repositórios de terceiros (que somente a segunda versão do algoritmo teria identificado). Curiosamente, os entrevistados 2, 9 e 11 tiveram, em seus perfis, mais *frameworks* descobertos que suas respostas. Durante a entrevista, esta questão foi perguntada ao Entrevistado 2. Ele afirmou que a presença de código Laravel deve-se à época em que estava estudando PHP e pode ter

utilizado alguma biblioteca do *software*. Esta resposta demonstra que, apesar de o usuário não ter conhecimento do *framework* em si, ele já trabalhou com, ao menos, alguma parte do mesmo. Esta mesma hipótese pode ser utilizada para as demais ocorrências. Ou ainda, os respondentes podem não se considerar familiarizados apesar de já terem trabalhado com eles.

O Entrevistado 10 também foi questionado a respeito de ter respondido "AngularJS e React", mas seu perfil possui apenas AngularJS. De acordo com ele, há repositórios em que trabalhou com React, mas estes seriam repositórios públicos que lhe garantiram acesso para *uploads* diretos (caso semelhante às contribuições do Entrevistado 5, citados anteriormente). Assim, este código não foi acessado pelo algoritmo e a ocorrência da ferramenta não foi disposta em sua análise.

No Quadro 13 faz-se uma comparação entre os *frameworks* respondidos pelos usuários e os que foram encontrados pela versão final do algoritmo. Neste quadro, exclui-se os *frameworks* respondidos como "Outros", uma vez que estes não fazem parte do escopo da busca deste trabalho. Assim, é possível visualizar de forma mais clara que a assertividade do protótipo foi, de forma geral, correta para a maioria das entrevistas.

Quadro 13 - Comparação entre os *frameworks* dos usuários e os encontrados

Entrevistado	Resposta dada	Versão final
Entrevistado 1	-	-
Entrevistado 2	AngularJS; React; Rails	Angular; Rails; Laravel; React
Entrevistado 3	AngularJS; Rails; Spring Framework	Rails; AngularJS
Entrevistado 4	AngularJS; React; Laravel, Rails	-
Entrevistado 5	-	-
Entrevistado 6	AngularJS	-
Entrevistado 7	AngularJS; Spring Framework	Spring Framework
Entrevistado 8	AngularJS; React; Rails; Spring Framework	AngularJS; React; Rails
Entrevistado 9	Laravel	Laravel; Symfony
Entrevistado 10	AngularJS; React	AngularJS
Entrevistado 11	React; Rails	Rails; React; AngularJS
Entrevistado 12	AngularJS; React; Rails; Spring Framework	Rails; AngularJS; React
Entrevistado 13	Rails; Volt	Rails

Fonte: elaborado pelo autor

Dessa forma, é difícil afirmar se os desenvolvedores em questão possuem, ou não, conhecimento da ferramenta. Se por uma perspectiva eles afirmam não ter, por outra eles podem ter trabalhado com as mesmas sem saber. Em um cenário de seleção de profissionais, este caso seria decidido através da subjetividade do analista. Outro ponto importante a ser destacado, é o fato de *frameworks* citados como "outros" (caso dos entrevistados 1, 4 e 7) são impossíveis de serem encontrados pelo protótipo, uma vez que não fazem parte do escopo do estudo, como explicado no Capítulo 2.

6.3. Conclusão dos resultados

Uma análise conclusiva a respeito do perfil de um programador baseando-se em suas contribuições no Github mostrou-se imprecisa. As características envolvidas na análise são, de fato, subjetivas e não seria possível obter uma conclusão correta para todos os analisados.

Os problemas envolvidos são vários: a não utilização da plataforma, a utilização da plataforma apenas para poucos projetos ou projetos específicos sem mais atividades, utilização ativa do Github em repositórios privados, o que torna impossível a rastreabilidade de informações. Outro cenário encontrado foi o de utilização ativa da plataforma em projetos públicos e projetos privados, tornando a análise ineficiente, pois não é possível processar todos os dados do usuário.

Porém, as análises se mostram úteis quando o relatório gerado é utilizado como ferramenta de apoio a tomada de decisão. Isto porque a ideia de ordenação das familiaridades de linguagens de programação é descartada e entende-se que todas as linguagens encontradas podem refletir certo conhecimento, apesar de não hierarquizado.

Outro ponto importante é que foi possível encontrar, para quase todos os casos, as linguagens com as quais os entrevistados trabalham no Github. Apesar de esta informação não refletir, com certeza, o perfil do analisado, mostra que as informações trazidas pelo *software* são acertadas. O protótipo, então, seria capaz de dizer corretamente com quais linguagens o usuário analisado trabalha no Github.

Este cenário mostra que a ideia de um *software* capaz de analisar perfis acertadamente para todos os casos é difícil. Para trabalhos futuros, é possível sugerir que dados encontrados no Github sejam cruzados com outras plataformas e redes sociais, a fim de

cruzar informações entre elas. O protótipo desenvolvido, no entanto, pode servir como sistema de apoio à seleção de candidatos.

6.4. Resumo do capítulo

Neste capítulo, os resultados das entrevistas foram apresentados e discutidos. Os dados dos questionários foram apresentados e, com eles, a comparação com os resultados obtidos a partir do protótipo. Também, mostrou-se as respostas dadas em entrevistas realizadas posteriormente a aplicação do questionário com três entrevistados a respeito dos dados encontrados no Github.

Assim, concluiu-se o objetivo primário deste trabalho, ou seja, entender se é possível utilizar o Github para identificar o perfil dos usuários. Como descreveu-se, devido à grande subjetividade em torno do que cada um define como conhecimento e, até mesmo, seu grau de familiaridade, não é possível obter uma conclusão final uniforme a todos. No entanto, quando compara-se as linguagens que os usuários dizem utilizar no Github e os dados provenientes do protótipo, é possível dizer que há assertividade nos resultados.

Discutiu-se, ainda, a identificação dos *frameworks*, parte importante na identificação dos ecossistemas. Quanto à eles, é possível afirmar que os resultados, para aqueles que utilizam o Github para hospedagem de projetos utilizando determinado *framework* foi satisfatório, demonstrando que esta funcionalidade do protótipo agiu como o esperado.

O objetivo geral desta pesquisa, que é "**Oferecer à administradores e analistas de RH um protótipo de *software* que colete informações a respeito de desenvolvedores candidatos, levando em consideração o perfil profissional desejado e as características dos profissionais que estão registradas no Github**", foi atendido, uma vez que é possível utilizar os dados providos pelo protótipo como sistema de apoio em uma seleção, tendo em vista que não é possível obter uma resposta final sobre nenhum deles.

CONSIDERAÇÕES FINAIS

As plataformas de codificação social apresentam-se hoje como verdadeiras redes sociais, possibilitando o desenvolvimento de *software* em conjunto com desenvolvedores de todas as partes do mundo. O foco principal foi no Github, plataforma deste trabalho de conclusão de curso. Sobre o Github, explicou-se a respeito de seu funcionamento, características e o meio de extração de dados. Esta extração dá-se através da API RESTful do Github, motivo pelo qual, apresentou-se o histórico da arquitetura REST e as características as quais *softwares* são arquitetados desta maneira. O Github, um *software* assim concebido, disponibiliza seus dados através de requisições GET feitas à sua API. Alguns exemplos destas requisições também foram apresentados.

Outro ponto apresentado foi o método utilizado para a escolha das linguagens de programação. Esta deu-se pelo serviço de busca de repositórios do Github, considerando as linguagens que possuem o maior número de repositórios estrelados. Este método também foi utilizado para a seleção dos *frameworks*. Para a escolha dos mesmos, buscou-se aqueles que estão hospedados no Github e possuíssem mais estrelas. Deste modo, a seleção deu-se pelas linguagens e *frameworks* mais populares no Github.

Utilizando o método descrito, dez *frameworks* foram escolhidos, dois de cada uma das linguagens mais populares no Github, para serem detectados pelo protótipo. Os *frameworks* selecionados foram: AngularJs e React (JavaScript); Laravel e Symfony (PHP); Rails e Volt (Ruby); Spring e LibGDX (Java); e Flask e Django (Python).

O protótipo foi desenvolvido utilizando diversas tecnologias, como o *framework* Ruby on Rails e a API do Github, que juntos mostram-se extremamente úteis e versáteis. Serviços de geolocalização, banco de dados, e biblioteca fornecida pelo Github, possibilitaram um desenvolvimento rápido e fácil, que permitiu focar, principalmente, no algoritmo desenvolvido e nos experimentos realizados.

Ao desenvolver o algoritmo, percebeu-se que o maior desafio era a descoberta de *frameworks*. Como esta informação não é fornecida pelo Github, e não há uma forma genérica de identificação para todos os *frameworks*, encontrou-se, nos gerenciadores de pacotes, uma forma de identificação. Devido à dependência da declaração destes nos gerenciadores de pacotes, este método pode apresentar falhas quando um programador decide utilizá-los de

forma não convencional. No entanto, as análises realizadas nos experimentos apresentaram resultados satisfatórios, demonstrando que o método utilizado pelo algoritmo é, geralmente, eficaz.

Outra possibilidade de utilização do protótipo foi a busca de profissionais por localidade. Os usuários da rede são analisados ao terem seus perfis vinculados à localidade pesquisada. O algoritmo utilizado é diferente do algoritmo utilizado para análises individuais. Estas diferenças foram listadas e explicadas, e resumem-se a não exibição de organizações, busca por linguagens principais de repositórios e exclusão de busca por contribuições públicas. O maior desafio desta busca deu-se em função do limite de consultas à API do Github, imposto pela própria rede social. Para análises de poucos usuários, as 5000 requisições são suficientes, porém, para grandes centros urbanos, foi preciso dividir a análise em intervalos de uma hora, para que novas requisições fossem possíveis. Ainda, o número de usuários buscados está restrito às primeiras 1000 pessoas encontradas, o que impossibilita a análise completa de grandes centros urbanos.

Também querer-se-ia verificar a assertividade do algoritmo nesta análise. Ao finalizar o protótipo, então, realizou-se a aplicação de uma pesquisa para validar sua eficácia. Primeiramente, um grupo de pré-teste, composto por 13 pessoas, responderam às questões e expuseram falhas na análise de perfis e dificuldade de comparação entre algumas respostas e a saída do protótipo. Por estes motivos, alterações foram realizadas no formulário e melhorias no algoritmo. Além disso, a metodologia utilizada abrangeu entrevistas ocorridas após a análise dos perfis. Estas foram realizadas para maior elucidação das respostas.

Os resultados expostos no capítulo 6 corroboraram o problema de pesquisa, que supunha ser possível utilizar análise automatizada para auxiliar no problema de seleção de possíveis candidatos. Algumas respostas dadas foram especialmente importantes para a aferição da hipótese sobre o Github ser utilizado como ferramenta de portfólio e currículo. Corroborando a pesquisa apresentada por Dabbish et al. (2012), diversas respostas fizeram referência a currículo, portfólios e visibilidade por parte do mercado.

Também percebeu-se, no capítulo de validação, que, apesar de ser possível atingir o objetivo de analisar automaticamente o perfil de programadores, existem problemas com afirmações finais a respeito de suas características. O protótipo consegue identificar, em sua maioria, os ecossistemas com os quais os profissionais têm familiaridade. Porém, a falta de

contribuições públicas e a subjetividade ao hierarquizar conhecimentos apresentam-se como fatores determinantes para impossibilitar a certeza na conclusão de cada análise.

Apesar disso, é importante destacar que a grande maioria das linguagens com as quais os entrevistados trabalham foram encontradas. Apenas um entrevistado, que não possui nenhuma contribuição pública, não obteve nenhum resultado. A identificação de *frameworks*, neste sentido, foi especialmente satisfatória, uma vez que não havia um modelo estabelecido para tal processo. As linguagens de programação, ainda que não haja assertividade em suas ordenações, foram, de modo geral, reconhecidas.

Como proposta de desenvolvimento futuro, considera-se que uma análise mais precisa é possível a partir do cruzamento dos dados do Github com outras plataformas. O Stack Overflow e o LinkedIn, juntamente com o Facebook, apresentam-se como possibilidades de averiguar se os dados vindos do Github são coincidentes. No entanto, não se tem informação a respeito dos termos destas redes sociais quanto a utilização de seus dados para este fim. Tal pesquisa seria necessário para prosseguir com a otimização do algoritmo apresentado aqui.

Outra possibilidade para otimização do algoritmo, para ser testada futuramente, seria parametrizar a busca realizada pelo algoritmo. Hoje, o filtro por *frameworks* é feito após a conclusão da busca por localidade, como apresentado no Capítulo 3. No entanto, é possível adicionar este filtro no momento da submissão da localidade, e ao executar o algoritmo, excluir a possibilidade dos *frameworks* não interessantes à busca. Desta forma, poupar-se-ia requisições utilizando apenas as classes referentes aos *frameworks* inseridas no filtro.

REFERÊNCIAS BIBLIOGRÁFICAS

ABES - ASSOCIAÇÃO BRASILEIRA DAS EMPRESAS DE SOFTWARE (Org.). **Mercado Brasileiro de Software**: panorama e tendências. Abes Software, São Paulo, p. 01-24, junho 2015. Disponível em: <http://central.abessoftware.com.br/Content/UploadedFiles/Arquivos/Dados 2011/ABES-Publicacao-Mercado-2015-digital.pdf>. Acesso em fevereiro de 2016.

ACIOLI, S. **Redes Sociais e Teoria Social:** Revendo os Fundamentos do Conceito. Informação & Informação, [S.l.], v. 12, n. 1esp, 2007. Disponível em: <http://www.uel.br/revistas/uel/index.php/informacao/article/view/1784>. Acesso em fevereiro de 2016.

BOWLER, T.; BANCER, W. **Symfony 1.3 Web Application Development**. 1a Ed. ed Birmingham: Packt Publishing Ltd, 2009.

CHACON, S.; STRAUB, Ben. **Pro Git**. 2a Edição. Nova Iorque: Apress, 2014. Disponível em <https://git-scm.com/book/en/v2>. Acesso em abril de 2016.

CHIAVENATO, I. **Gestão de Pessoas:** O novo papel dos recursos humanos nas organizações. 4a edição. Barueri, SP: Manole, 2014.

COLLINS-SUSSMAN, B.; FITZPATRICK, B. W.; PILATO, C. M. **Version Control with Subversion:** For Subversion 1.7. Stanford: 2013. Disponível em: <http://svnbook.red-bean.com/en/1.7/svn-book.pdf>

DABBISH, L.; STUART, C.; TSAY, J.; HERBSLEB, J. **Social Coding in GitHub:** Transparency and Collaboration in an Open Software Repository. 2012. Curso de Ciências da Computação, School Of Computer Science And Center For The Future Of Work, Heinz College Carnegie Mellon University, Pitsburgo, 2012. Disponível em: <https://www.cs.cmu.edu/~xia/resources/Documents/cscw2012_Github-paper-FinalVersion-1.pdf>. Acesso em fevereiro de 2016.

EXAME (Org). **Mercado brasileiro de TI cresceu mais de 15% em 2013**. 2014. Disponível em: <http://exame.abril.com.br/tecnologia/noticias/mercado-brasileiro-de-ti-cresceu-mais-de-15-em-2013>. Acesso em fevereiro de 2016.

FACEBOOK INC (Org). A JavaScript Library for Building User Interfaces. **React**, 2016. Disponível em: <https://facebook.github.io/react/>. Acesso em Maio de 2016.

FIELDING, R. T. **Representational State Transfer (REST)**. 2000. Tese (Doutorado em Filosofia na Informação e Ciência da Computação) - Universidade da Califórnia, Irvine, 2000. Disponível em: < http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>. Acesso em abril de 2016.

FLANAGAN, D.; MATSUMOTO, Y. **The Ruby Programming Language**. 1a Edição. ed. Sebastopol: O'Reilly Media, 2014. cap. 1, p. 1-24.

FOLHA DE SÃO PAULO (Org.). **Faltam 45 mil profissionais de TI no Brasil**. 2014. Disponível em <http://classificados.folha.uol.com.br/empregos/2014/06/1466085-faltam-45-mil-profissionais-de-ti-no-brasil.shtml>. Acesso em março 2016.

FOWLER, M. **InversionOfControl**. 2005. Disponível em <http://martinfowler.com/bliki/InversionOfControl.html>. Acesso em maio 2016.

GEER, D. **Will Software Developers Ride Ruby on Rails to Success?**. Technology News, IEEE. 2006.

GITHUB DEVELOPER (Org.). **Github API V3**. 2016. Disponível em <https://developer.github.com/v3/>. Acesso em maio 2016.

GITHUB HELP (Org.). **Github Help**. 2016. <https://help.github.com/articles/search-syntax/>. Acesso em maio de 2016.

GLOBO NEWS (Org.). **Empresas investem no treinamento de lideranças para melhorar resultados**. 2015. Disponível em: <http://g1.globo.com/globo-news/contacorrente/noticia/2015/01/empresas-investem-no-treinamento-de-liderancas-para-melhorar-resultados.html>. Acesso em fevereiro de 2016.

GRINBERG, M. **Flask Web Development**. 1a Edição. ed. Sebastopol: O'Reilly Media, 2014.

HOLOVATY, A.; KAPLAN-MOSS, J. **The Definitive Guide to Django**: Web Development Done Right. 2a Edição. ed Berkley: Apress, 2009.

JOHNSON, R. E.; FOOTE, Brian. **Designing Reusable Classes**. Journal of Object-Oriented Programming. 1988. Departamento de Ciência da Computação, Universidade de Illinois. Disponível em <http://www.cse.msu.edu/~cse870/Input/SS2002/MiniProject/Sources/DRC.pdf>. Acesso em maio de 2016.

JORNAL DA GLOBO (Org.). **Mercado de TI é um dos setores que não pararam de contratar no Brasil.** Disponível em < http://g1.globo.com/jornal-da-globo/noticia/2016/02/mercado-de-ti-e-um-dos-setores-que-nao-pararam-de-contratar-no-brasil.html>. Acesso em março 2016.

KALLIAMVAKOU, E.; SINGER, L.; GOUSIOS, G.; GERMAN, D., M.; BLINCOE, K.; DAMIAN, D. **The Promises and Perils of Mining GitHub**. ACM: Nova Iorque. 2014. p. 92-101. Disponível em <http://keg.cs.uvic.ca/pubs/kalliamvakou-MSR2014.pdf>. Acesso em abril de 2016.

KAYAL, D. **Pro Java EE Spring Patterns**. 1a Edição. ed Berkley: Apress, 2008.

LIMA, A.; ROSSI, L.; MUSOLESI, M. **Coding Together at Scale:** GitHub as a Collaborative Social Network. [Birmingham]: Universidade de Birmingham, 2014.

LOELIGER, J. **Version Control with Git**. 1a Edição. ed. Sebastopol: O'Reilly Media, 2009. Disponível em <http://www.foo.be/cours/dess-20122013/b/OReilly%20Version%20Control%20with%20GIT.pdf>. Acesso em março de 2016.

MACEDO, M. C. B. **O Mercado de Trabalho em Tecnologia de Informação: a inserção profissional dos desenvolvedores de software**. 2011. 103 f. Dissertação (Mestrado) - Curso de Sociologia, Instituto de Filosofia e Ciências Humanas, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2011. Disponível em: <https://www.lume.ufrgs.br/bitstream/handle/10183/49103/000827129.pdf?sequence=1>. Acesso em fevereiro de 2016.

MAGUIRE, J. **The SourceForge Story**. Datamation, [s.l.], 2007. Disponível em: <http://web.archive.org/web/20110804024950/http://itmanagement.earthweb.com/cnews/article.php/3705731>. Acesso em: 26 abr. 2016.

NAIR, S. B.; OEHLKE, A. **Learning LibGDX Game Development**. 2a Edição. ed Birmingham: Packt Publishing, 2015.

PRABHAKAR, B.; LITECKY, C.; ARNETT, K. IT Skills in a Tough Job Market. **Communications of the ACM**, v. 48, n. 10, p. 91-94, Outubro 2005.

PRODANOV, C. C.; FREITAS, E. C. **Metodologia do trabalho científico:** métodos e técnicas da pesquisa e do trabalho acadêmico, 2013. Disponível em: <http://www.feevale.br/cultura/editora-feevale/metodologia-do-trabalho-cientifico---2-edicao>. Acesso em fevereiro de 2016.

RECUERO, R. **Redes sociais na internet**. Porto Alegre: Sulina, 2009.

RICHARDSON, L.; RUBY, S. **RESTful Serviços Web**. Rio de Janeiro: Alta Books, 2007.

ROCHKIND, M. J. **The Source Code Control System**. IEEE Transactions On Software Engineering. [S.l.], v. 1, n. 4, p. 364-369, 1975.

RUPARELIA, N. B. **The History of Version Control**. ACM SIGSOFT Software Engineering Notes, Nova Iorque, v. 35, n. 1, p. 5-9, 2010.

SANATINIA, A.; NOUBIR, G. **On GitHub's Programming Languages**. Faculdade de Computação e Ciência da Informação, Northeastern University, Boston, 2016.

SAUNIER, R. **Getting Started with Laravel 4**. 1a Edição. ed. Birmingham: Packt Publishing Ltd, 2014.

SCHAAB, R. **Estudo Das Práticas De Gestão Do Conhecimento Em Atividades De Inovação De Produtos E Serviços Em Uma Empresa De Tecnologia Da Informação**. f. 222. 2016. Dissertação (Mestrado Em Indústria Criativa) - Universidade Feevale, Novo Hamburgo, RS, 2016. cap. 6, p. 81-140.

SEBESTA, R. W. **Conceitos de linguagens de programação**. 9. ed. Porto Alegre, RS: Bookman, 2011.

SESHADRI, S.; GREEN, B.. **AngularJS Up & Running**: Enhanced Productivity With Structured Web Apps. 1a Edição. ed. Sebastopol: O'Reilly Media, 2014.

SILVA, T. R. D. **PyRester**: Uma abordagem baseada em modelos U2TP para geração de código de teste unitário para RESTful Web Services. Universidade Federal do Rio Grande do Sul. Curso de Ciência da Computação, Porto Alegre, BR–RS, 2011. p. 28-34.

SOFTEX (Org.). **Relatório SOFTEX.** 2014. Disponível em: <http://www.softex.br/wp-content/uploads/2015/04/Relatorio_Anual_2014.pdf>. Acesso em março de 2016.

SOURCEFORGE (Org.). **About**. SourceForge, [s.l.], 2016. Disponível em: < https:// sourceforge.net/about>. Acesso em: 26 abr. 2016.

STOUT, R. Docs - Volt. **Volt**, 2016. Disponível em: <http://voltframework.com/docs>. Acesso em: Maio de 2016.

THUNG, F.; LO, D.; JIANG, L. **Network Structure of Social Coding in GitHub**. CSMR 2013: Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering: 5-8 Março 2013, Genova, Itália. Research Collection School Of Information Systems. Disponível em: <http://ink.library.smu.edu.sg/sis_research/1687>. Acesso em fevereiro de 2016.

TIBOE INDEX (Org.). **TIOBE Programming Community Index Definition**. 2016. Disponível em <http://www.tiobe.com/tiobe_index?page=programminglanguages_definition>. Acesso em maio de 2016.

VEPSÄLÄINEN, J. **SurviveJS Webpack and React**: From Apprentice to Master. 2a Edição. CreateSpace Independent Publishing Platform, 2016.

YOSHIKAWA, Y; IWATA, T.; SAWADA, H.**Collaboration on Social Media:** Analyzing Successful Projects on Social Coding. 2014.

APÊNDICE A – Questionário aplicado no pré-teste

Você está sendo convidado(a) a participar do TCC de graduação intitulado: ANÁLISE AUTOMATIZADA DE HABILIDADES DE DESENVOLVEDORES BASEADA EM CONTRIBUIÇÕES NO GITHUB. O trabalho está sendo realizado pelo acadêmico Eduardo Henrique Kasper do curso de Sistemas de Informação, orientado e co-orientado pelos professores Marta R. Bez e Juliano V. de Carvalho, respectivamente. O objetivo deste trabalho é criar um protótipo onde, ao informar o usuário no Github, analisará o perfil do mesmo, baseando-se em suas informações públicas. Por isso, as informações a respeito do entrevistado serão usadas exclusivamente para fins acadêmicos e não serão divulgadas.

* Obrigatório

1. Qual seu nome?

2. Qual seu email?

3. Qual seu usuário no Github *

4. Desde quando você possui conta no Github? *

5. Você se considera um usuário ativo na plataforma?

○ Sim ○ Não

6. Em qual cidade você reside? *

Por favor, selecione seu grau de familiaridade/fluência para com as linguagens de programação/marcação listadas, sendo 0 nenhuma e 10 muita familiaridade/fluência

7. PHP *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

8. JavaScript *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

9. Java *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

10. Ruby *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

11. Python *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

12. C *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

13. C# *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

14. Haskell *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

15. Clojure *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

16. Go *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

17. HTML *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

18. CSS *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

19. CoffeeScript *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

20. Lua *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

21. Matlab *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

22. Perl *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

23. R *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

24. Shell/Shell Script *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

25. Scala

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

26. Objective-C *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

27. VimL *

	0	1	2	3	4	5	6	7	8	9	10	
Nenhuma												Fluente

28. Por favor, assinale os *frameworks* que você domina ou tem conhecimento prático *

- [ ] JavaScript - AngularJS
- [ ] JavaScript - React
- [ ] PHP - Laravel
- [ ] PHP - Symfony
- [ ] Ruby - Rails
- [ ] Ruby - Volt
- [ ] Java - LibGDX
- [ ] Java - Spring Framework
- [ ] Python - Flask
- [ ] Python - Django
- [ ] Other: _____

29. Em quantos repositórios você já trabalhou no Github *

- ○ Entre 1 e 5
- ○ Entre 6 e 10
- ○ Entre 11 e 15
- ○ Entre 16 e 20
- ○ Mais de 20

30. Você acredita que usar o GITHUB lhe traz ou pode trazer alguma vantagem *

○ Sim ○ Não

31. Se sim, qual(is)? *

APÊNDICE B – Questionário aplicado no pós-teste

Você está sendo convidado(a) a participar do TCC de graduação intitulado: ANÁLISE AUTOMATIZADA DE HABILIDADES DE DESENVOLVEDORES BASEADA EM CONTRIBUIÇÕES NO GITHUB. O trabalho está sendo realizado pelo acadêmico Eduardo Henrique Kasper do curso de Sistemas de Informação, orientado e co-orientado pelos professores Marta R. Bez e Juliano V. de Carvalho, respectivamente. O objetivo deste trabalho é criar um protótipo onde, ao informar o usuário no Github, analisará o perfil do mesmo, baseando-se em suas informações públicas. Por isso, as informações a respeito do entrevistado serão usadas exclusivamente para fins acadêmicos e não serão divulgadas.

1. Qual seu nome?

2. Qual seu email?

3. Qual seu usuário no Github *

4. Desde quando você possui conta no Github? *

5. Você se considera um usuário ativo na plataforma?
○ Sim ○ Não

6. Se sim, qual motivo faz de você um usuário ativo?

7. Em qual cidade você reside? *

8. Em quantos repositórios você já trabalhou no Github *
○ Entre 1 e 5
○ Entre 6 e 10
○ Entre 11 e 15
○ Entre 16 e 20
○ Mais de 20

9. Você trabalha em repositório(s) privado(s) *
○ Sim ○ Não

10. Para quais fins você utiliza o Github? *

☐ Trabalho
☐ Estudo/Escola/Universidade
☐ Projetos Pessoais
☐ Contribuições em Projetos Open Source
☐ Other: _____

Por favor, ordene as linguagens abaixo de acordo com seu grau de familiaridade/fluência com elas. Deixe em branco para nenhuma familiaridade/fluência

Aqui, procura-se saber a ordem de familiaridade/fluência que você possui com as linguagens listadas. Marque 1 para a linguagem que você mais conhece, 2 para a segunda posição, etc.

11.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
PHP																			
JavaScript																			
Ruby																			
Java																			
Python																			
C																			
C#																			
Haskell																			
Clojure																			
CoffeeScript																			
Go																			
Lua																			
Matlab																			
Perl																			
R																			
Shell/Shell Script																			
Scala																			
Objective-C																			
VimL																			

Por favor, entre as linguagens que você conhece, ordene, da mesma forma feita anteriormente, as linguagens abaixo com as quais você costuma trabalhar no Github

12.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
PHP																			
JavaScript																			
Ruby																			
Java																			
Python																			
C																			
C#																			
Haskell																			
Clojure																			
CoffeeScript																			
Go																			
Lua																			
Matlab																			
Perl																			
R																			
Shell/Shell Script																			
Scala																			
Objective-C																			
VimL																			

13. Por favor, assinale os *frameworks* que você domina ou tem conhecimento prático *

- ☐ JavaScript - AngularJS
- ☐ JavaScript - React
- ☐ PHP - Laravel
- ☐ PHP - Symfony
- ☐ Ruby - Rails
- ☐ Ruby - Volt
- ☐ Java - LibGDX
- ☐ Java - Spring Framework
- ☐ Python - Flask
- ☐ Python - Django
- ☐ Other: _____

14. Você acredita que usar o GITHUB lhe traz ou pode trazer alguma vantagem *

○ Sim ○ Não

15. Se sim, qual(is)?

APÊNDICE C – Linguagens dos entrevistados e as obtidas pela primeira versão do algoritmo

	Linguagens ordenadas pelo usuário	Linguagens ordenadas pelo protótipo
Entrevistado 1	1. PHP 2. JavaScript 3. Java 4. C	1. Java
Entrevistado 2	1. JavaScript 2. Ruby 3. PHP 4. Java	1. Ruby 2. JavaScript 3. PHP 4. CSS 5. Java 6. Shell/ ShellScript 7. HTML 8. ApacheConf 9. Perl
Entrevistado 3	1. Ruby 2. Java 3. JavaScript 4. Clojure 5. C# 6. CoffeScript 7. Shell/ ShellScript 8. Python 9. C 10. Haskell 11. Lua 12. Matlab 13. Perl 14. Objective C	1. Ruby 2. Clojure 3. JavaScript 4. HTML 5. Java 6. TypeScript 7. CSS 8. CoffeeScript 9. Perl
Entrevistado 4	1. PHP 2. JavaScript 3. CoffeeScript 4. Ruby 5. Java 6. Python 7. Shell/Shell Script 8. VimL 9. Objective C 10. Scala 1. R 2. Perl 3. Matlab 4. Lua 5. Go 6. Clojure 7. Haskell 8. C# 9. C	1. PHP 2. JavaScript 3. Java 5. JupyterNotebook 6. CSS
Entrevistado 5	1. Java 2. C	-
Entrevistado 6	1. Java 2. JavaScript 3. PHP 4. C# 5. C 6. Shell/ShellScript	1. Java 2. JavaScript 3. CSS 4. HTML
Entrevistado 7	1. Java 2. JavaScript 3. Shell/ShellScript 4. Python	1. Java 2. Shell/ShellScript 3. HTML 4. CSS 5. Python
Entrevistado 8	1. JavaScript 2. Java 3. Python 4. Shell/ShellScript 5. Ruby 6. Objective C 7. C# 8. C	1. Javascript 2. CSS 3. Python 4. Java 5. C 6. HTML 7. C++ 8. Shell/ ShellScript
Entrevistado 9	1. PHP 2. JavaScript 3. Java 4. C	1. PHP 2. Java
Entrevistado 10	1. JavaScript 2. C# 3. Java 4. Ruby 5. Objective C 6. Perl	1. JavaScript 2. Java 3. C# 4. CSS 5. ERlang

Entrevistado 11	1. Ruby 2. PHP 3. JavaScript 4. Shell/ShellScript 5. Python 6. CoffeeScript 7. C# 8. C 9. Perl	10. Java 11. Go 12. Haskell 13. Clojure 14. Lua 15. Matlab 16. Scala 17. R 18. Objective-C 19. VimL	1. JavaScript 2. Ruby 3. HTML	4. CSS 5. PHP 6. Go
Entrevistado 12	1. Ruby 2. Java 3. JavaScript	4. CoffeeScript 5. ShellScript	1. Ruby 2. JavaScript 3. Shell	
Entrevistado 13	1. Ruby 2. C 3. CoffeeScript 4. JavaScript	5. PHP 6. Java 7. Python 8. Shell/ShellScript	1. Crystal 2. Ruby 3. JavaScript 4. Shell/ShellScript 5. HTML	6. M 7. C 8. CSS 9. Matlab 10. Standard ML

APÊNDICE D – Linguagens inseridas pelos entrevistados e as obtidas pela segunda versão do algoritmo

	Linguagens ordenadas pelo usuário	Linguagens ordenadas pelo protótipo	
		Repositórios do Usuário	Contribuições públicas
Entrevistado 1	1. PHP 2. JavaScript 3. Java 4. C	1. Java 2. JavaScript 3. CSS	
Entrevistado 2	1. JavaScript 2. Ruby 3. PHP 4. Java	1. PHP 2. JavaScript 3. CSS 4. Ruby 5. Shell/ShellScript	
Entrevistado 3	1. Ruby 2. Java 3. JavaScript 4. Clojure 5. C# 6. CoffeScript 7. Shell/ ShellScript 8. Python 9. C 10. Haskell 11. Lua 12. Matlab 13. Perl 14. Objective C	1. HTML 2. Ruby 3. JavaScript 4. TypeScript	Ruby - Clojure - HTML - JavaScript - CSS - CoffeeScript
Entrevistado 4	1. PHP 2. JavaScript 3. CoffeeScript 4. Ruby 5. Java 6. Python 7. Shell/Shell Script 8. VimL 9. Objective C 10. Scala 1. R 2. Perl 3. Matlab 4. Lua 5. Go 6. Clojure 7. Haskell 8. C# 9. C	1. CSS 2. Jupyter Notebook 3. HTML 4. PHP 5. JavaScript 6. Python 7. Batchfile 8. Makefile 9. Java	1. PHP 2. JavaScript 3. CSS
Entrevistado 5	1. Java 2. C	-	-
Entrevistado 6	1. Java 2. JavaScript 3. PHP 4. C# 5. C 6. Shell/ShellScript	1. JavaScript 2. Java 3. CSS 4. HTML	-
Entrevistado 7	1. Java 2. JavaScript 3. Shell/ShellScript 4. Python	1. Java 2. Python 3. HTML 4. CSS	1. Java 2. JavaScript 3. CSS 4. SQLPL
Entrevistado 8	1. JavaScript 2. Java 3. Python 4. Shell/ShellScript 5. Ruby 6. Objective C 7. C# 8. C	1. C 2. JavaScript 3. C++ 4. Java 5. Python 6. CSS 7. HTML	1. C++ 2. Python 3. JavaScript 4. Ruby 5. HTML 6. CSS 7. CoffeeScript 8. Groovy 9. Objective C

Entrevistado 9	1. PHP 2. JavaScript	3. Java 4. C	1. PHP 2. Java 3. HTML	
Entrevistado 10	1. JavaScript 2. C# 3. Java	4. Ruby 5. Objective C 6. Perl	1. CSS 2. JavaScript 3. Java 4. C# 5. HTML 6. Erlang	1. C#
Entrevistado 11	1. Ruby 2. PHP 3. JavaScript 4. Shell/ShellScript 5. Python 6. CoffeeScript 7. C# 8. C 9. Perl	10. Java 11. Go 12. Haskell 13. Clojure 14. Lua 15. Matlab 16. Scala 17. R 18. Objective-C 19. VimL	1. JavaScript 2. Ruby 3. CSS 4. C# 5. HTML 6. PHP 7. Elixir	1. Ruby 2. JavaScript 3. HTML 4. C# 5. CSS
Entrevistado 12	1. Ruby 2. Java 3. JavaScript	4. CoffeeScript 5. ShellScript	1. JavaScript 2. Ruby 3. HTML 4. Shell/ShellScript	1. Ruby 2. Shell/ ShellScript 3. HTML 4. Java 5. C 6. CSS 7. JavaScript 8. Cucumber
Entrevistado 13	1. Ruby 2. C 3. CoffeeScript 4. JavaScript	5. PHP 6. Java 7. Python 8. Shell/ShellScript	1. Crystal 2. Matlab 3. Ruby 4. JavaScript 5. HTML 6. Shell/ShellScript 7. C	1. Crystal 2. Ruby 3. Shell/ ShellScript 4. VimL