UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANGELO PEREIRA GHEZZI

API para auxílio de mineração de repositórios Git e SVN

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Marcelo Soares Pimenta

Porto Alegre
2015

AGRADECIMENTOS

Agradeço a todos que conheci durante a jornada até este ponto. Querendo ou não todas as experiências contribuem de certa forma no que nos tornamos.

RESUMO

Este trabalho propõe uma API que auxilia na extração de código em tipos diferentes de repositórios. A quantidade de código aberto disponível online é enorme e existem diversos acervos de projetos online. Com a existência de sistemas de controle de versão existem dados que possibilitam traçar o progresso do desenvolvimento destes projetos. Com a mineração de todos estes dados está a possibilidade de descobrir diversos aspectos sobre o desenvolvimento de software. Já existem várias pesquisas focadas no que pode ser descoberto e analisado a partir da mineração de repositórios. Mas nem sempre os sistemas de controle de versão apresentam estruturas semelhantes e não existe um acesso universal para todos os tipos diferentes. A capacidade de coletar os dados de repositórios sem se preocupar com o sistema de controle de versão que estes usam aumenta a quantidade de projetos que uma pesquisa pode ter acesso. Visto que existem pontos em comum nos tipos de informação que são gerados por diferentes sistemas, a criação de uma camada que esconde as suas diferentes implementações pode ser formada. Com este trabalho criamos então uma API capaz de realizar a busca de projetos que utilizam sistemas de controle de versões Git ou SVN seguido da extração da lista de branches e tags criadas por estes. Ela é estruturada de forma que uma camada de abstração permite que seu uso para os diferentes sistemas de controladores de versão é feito de uma mesma maneira independente das diferenças entre eles. Para demonstrar seu uso foi desenvolvida uma ferramenta que integra os elementos da API.

Palavras-chave: Sistemas de controle de versões. Busca de repositórios. Mineração de repositórios. Git. Subversion.

API to aid with Git and SVN repository mining

ABSTRACT

This paper proposes an API that helps with code extraction from different types of repository. The quantity of open source code available online is enormous and several online project collections exist. The use of version control systems brings the creation of data which allows for the tracking of the development progress of these projects. Data mining all that data gives researchers the possibility of looking into several aspects of software development. There are already several works that focus on what can be discovered and analyzed from mining repositories that use version control systems. However these version control systems do not always follow the same structure and there is no universal access method for different types. Being able to collect the data from repositories without having to worry about the version control system being used increases the number of projects a research has access to work with. Focusing on the similar types of information generated by different systems it is possible to develop a layer that abstracts the implementation differences from version control systems. In this paper we developed an API capable of conducting a search for projects that use the Git or SVN version control systems followed by the extraction of the list of branches e tags generated by them. It is designed so that an abstraction layer allows using different version control systems with the same method regardless of their differences. A tool that integrates the elements of the API was developed to demonstrate it.

Keywords: Version Control Systems. Repository Searching. Repository Mining. Git. Subversion.

LISTA DE FIGURAS

Figura 5.1 – Diagrama UML completo da API.....18
Figura 5.2 – Diagrama UML da API ampliado no canto superior esquerdo.....19
Figura 5.3 – Diagrama UML da API ampliado no canto superior direto.....23
Figura 5.4 – Diagrama UML da API ampliado na metade inferior.....26
Figura 5.5 – Diagrama de Sequência.....27
Figura 5.6 – Captura de tela da demonstração da ferramenta.....29
Figura 5.7 – Captura de tela da demonstração da ferramenta.....29
Figura 5.8 – Captura de tela da demonstração da ferramenta.....30

LISTA DE TABELAS

Tabela 4.1 – Comparação entre os trabalhos analisados.....17

LISTA DE ABREVIATURAS E SIGLAS

API	Application Program Interface
CVS	Concurrent Versions System
FOSS	Free and Open Source Software
SCM	Software Configuration Management
SVN	Subversion
URL	Uniform Resource Locator
VCS	Version Control System

SUMÁRIO

1 INTRODUÇÃO....9
2 FUNDAMENTAÇÃO TEÓRICA....10
2.1 Sistemas de Controle de Versão....10
2.1.1 Git....10
2.1.2 Subversion....11
2.1.3 Branches e Tags....11
2.2 Busca de Projetos....12
2.3 Mineração de Repositórios....12
3 O PROBLEMA....13
3.1 Motivação do Problema....13
3.2 Definição do Problema....13
4 ESTADO DA ARTE....14
4.1 Sourcerer....14
4.2 GHTorent....15
4.3 Rminer....15
4.4 Análise Comparativa....16
5 A PROPOSTA....18
5.1 Busca de Projetos....19
5.1.1 GitHubAggregator....20
5.1.2 OpenHubAggregator....21
5.2 Acesso ao Controlador de Versão....23
5.2.1 GitAccess....24
5.2.2 SVNAccess....25
5.3 Project, VersionTree e a interação do todo....25
5.3.1 Project e VersionTree....25
5.3.2 Funcionamento da API....27
5.3.3 Ferramenta que utiliza a API....28
6 CONCLUSÃO....33
REFERÊNCIAS....34

1 INTRODUÇÃO

A quantidade de software de livre acesso e facilmente encontrado na internet cresce cada dia mais. A capacidade de conseguir coletar parte deste vasto conhecimento para análise abre a possibilidade de gerar avanços nas soluções da Engenharia de Software. Quanto maior o volume do conhecimento existente, maior a necessidade de ferramentas que sejam capazes de obter a fração que interessa a um certo problema de modo que a análise dos códigos já existentes tenha um rumo. Quando queremos olhar para as alterações que um código sofre ao longo de seu desenvolvimento, o uso de um sistema de controle de versão possui um histórico destas alterações. Com este histórico é possível analisar o progresso do código. Este trabalho é feito com o intuito de criar uma API capaz de obter os *branches* e *tags* de um projeto auxiliando na extração de código em tipos diferentes de repositórios. A API tem então o objetivo de abstrair o processo de encontrar projetos e ter como resultado a lista de *branches* e *tags*, adicionado das informações que identificam o projeto.

O texto começa no capítulo 2 com a introdução e explicação dos conceitos que contribuem para a definição do problema. No capítulo 3 o problema é definido. Em seguida no capítulo 4 são apresentados alguns trabalhos com relevância ao nosso e uma comparação com a solução proposta. No capítulo 5 são discutidos os detalhes da solução proposta e apresentamos uma ferramenta demonstrando a sua utilização. O texto é concluído no capítulo 6.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentada uma breve introdução sobre sistemas de controle de versão, busca por projetos existentes, mineração de repositórios, e também a motivação e definição do problema.

2.1 Sistemas de Controle de Versão

Neste texto, repositório irá se referir a um repositório de código-fonte mantido em um sistema de controle de versão, onde desenvolvedores podem cooperar no desenvolvimento de um software. Uma cópia de um repositório é em si um repositório e conforme será mencionado ao longo deste capítulo esta cópia pode ser chamada de *branch*. Um *branch* é uma linha de desenvolvimento que existe independentemente de outra linha (B. Collins-Sussman et al., 2004). Ou seja, quando há uma divergência entre duas cópias do projeto.

Um sistema de controle de versão (ou versionamento), VCS (do inglês *version control system*) ou ainda SCM (do inglês *source code management*) é um software empregado com o objetivo de gerenciar diferentes versões de um projeto. Como eles são usados na prática difere de uma entidade para outra (A. Bachmann e A. Bernstein, 2009). Alguns dos VCS mais comuns são: *Subversion (SVN), Concurrent Version System (CVS), Mercurial* e *Git.* Seguiremos com um foco no *Git* e no *SVN.*

2.1.1 Git

Git é um sistema de controle de versão distribuído. Toda informação sobre um repositório é guardada localmente e não em um servidor remoto (T. Zimmermann et al., 2005). O *Git* possui uma ênfase em velocidade, integridade dos dados e suporte para *workflows* não lineares e distribuídos. Hoje ele pode ser considerado um dos VCS mais utilizado por desenvolvedores de software atingindo uma grande aderência desde sua criação em 2005 (S. Chacon e B. Straub, 2009).

Conforme J. Loeliger e M. McCullough (2012) já existem diversas ferramentas para Git mas o GitHub se destaca entre elas. Ele é em sua essência um site onde podem ser hospedados repositórios porém ele retém um aspecto que pode ser comparado ao de redes sociais ao permitir um acompanhamento do que outros usuários estão realizando com seu acesso livre para alterações em um código sem afetar a cópia principal.

2.1.2 Subversion

Subversion é um sistema de controle de versão centralizado. Ele conta com um repositório central semelhante a um sistema de arquivos, porém com a capacidade de lembrar todas as alterações feitas em seus arquivos e diretórios (B. Collins-Sussman et al., 2004). Com a capacidade de acessar um repositório através de redes é possível que diversos colaboradores trabalhem em um mesmo conjunto de dados. Visto que o progresso é versionado ao longo do desenvolvimento é possível trabalhar com a segurança de que qualquer problema pode ser revertido. Porém essa dependência a um servidor dificulta o uso do sistema quando não há uma conexão de rede.

No sistema *SVN* é normal que uma alteração só seja feita no repositório após a mesma ser vetada, enquanto o trabalho individual dos colaboradores é em grande parte invisível para o sistema (C. Bird et al., 2009). Enquanto que no *Git* as alterações não são feitas diretamente em um repositório central e sim em um repositório local. Os colaboradores podem trabalhar independentemente e unir seus progressos apenas quando desejarem. Observa-se então que é fácil atribuir números às alterações em um repositório *SVN* de forma incremental já que elas são feitas de forma sequencial. O mesmo não acontece com *Git* já que as alterações feitas localmente podem ocorrer em paralelo e portanto são utilizados identificadores *SHA-1*.

2.1.3 Branches e Tags

Durante o desenvolvimento de um projeto usando um repositório é possível a criação de *branches* e *tags*. Uma *tag* em ambos os *Git* e *SVN* tem o propósito de ser uma cópia do repositório no momento de sua criação. Elas são criadas manualmente e geralmente com o intuito de marcar uma *release* ou uma etapa importante do projeto.

Um *branch* é criado no momento em que um colaborador copia um repositório e começa a realizar alterações em sua cópia que não estão sendo refletidas no original com o propósito de seguir uma linha de desenvolvimento diferente. No *SVN* funciona como se fosse criada uma cópia de todo o conteúdo de um diretório em um novo diretório enquanto que no *Git* é como se fosse salvo um novo ponteiro que faz referência ao nodo em que foi criado e é livre para seguir um caminho diferente (S. Chacon e B. Straub, 2009).

2.2 Busca de Projetos

Com o número enorme de projetos existentes existe a necessidade de filtrar um acervo de modo a encontrar apenas aqueles que nos interessam. Já existem ferramentas de busca online que nos permitem realizar esta tarefa. Entre as existentes destacamos a API do *GitHub* que contém métodos para tal. Porém apenas para projetos hospedados no mesmo, uma quantia considerável que já ultrapassou 10 milhões de repositórios.

Também é dada atenção ao *OpenHub*, um site que indexa projetos de software que é capaz de pesquisar por códigos na internet e encontrar resultados para pelo menos 5 VCS diferentes. Em sua *homepage* (2015) ele se considera uma comunidade online e um diretório público de *FOSS (free and open source software).* Ele oferece dados analíticos e serviços de busca para descobrir, avaliar, acompanhar e comparar projetos. Indexando mais de 21,000,000,000 de linhas de código e com um sistema *wiki* onde é possível que qualquer um adicione ou corrija páginas de projetos. Isso permite que o *OpenHub* seja um dos maiores, mais preciso e mais atualizado diretório *FOSS* disponível.

2.3 Mineração de Repositórios

Mineração de repositórios é o processo de coletar e analisar os dados referentes ao conteúdo desses repositórios (W. Shahbazian, 2010). Esses dados podem ser então utilizados para analisar os códigos e dependendo do tipo de informação coletada é possível obter diversas métricas como por exemplo a complexidade ciclomática de um código que indica a complexidade do software através da quantidade de caminhos de execução independentes. Essa métrica desenvolvida por T.J.Mcabe (1976) seria apenas um exemplo do que pode ser obtido a partir da mineração de um repositório.

Como apresentamos anteriormente existem diversos tipos de VCS e a organização deles não segue um padrão único. Devido a isso a maneira para que se possa obter informação, ou até mesmo a forma em que a mesma se apresenta, de um repositório pode variar de um VCS para outro.

3 O PROBLEMA

3.1 Motivação do Problema

Extrair informações de repositórios não é uma tarefa trivial devido à presença de diferenças entre eles. Mas mesmo assim existe uma vantagem em conseguir abranger o máximo possível do vasto conhecimento já existente e livremente disponível online. Quando estamos interessados em extrair os dados de diversos projetos relacionados por algum aspecto em comum mas que utilizam VCS distintos esta barreira precisa ser quebrada.

Mineração de repositórios, quando feita manualmente, é algo tedioso. A automatização desta tarefa repetitiva é algo desejável que deve ser útil para àqueles que interessam. E ainda mais desejável seria uma ferramenta que permita a possibilidade de abstrair tal extração de informação de VCS distintos em uma experiência única.

3.2 Definição do Problema

Neste trabalho visamos então desenvolver uma API capaz de, a partir de palavras chaves, a busca por projetos (Git ou SVN) que as contenham em seu título seguido da extração, feita sem clonar os repositórios, da lista de *branches* e *tags* presentes neste junto de alguns dados básicos para identificação dos projetos. O objetivo da API é auxiliar na coleta de projetos para o uso desses em pesquisas acadêmicas.

Procuramos desenvolver esta API de uma maneira que ela consiga ser incrementada posteriormente para agir ser capaz de acessar outros tipos de VCS, mantendo uma camada de abstração acima das suas implementações específicas.

4 ESTADO DA ARTE

Ao pesquisar na internet por trabalhos anteriores com relevância ao tratado por este verificamos que já existem ferramentas que extraem informações de repositórios Git, assim como existem ferramentas que mineram repositórios de outros VCS. Parte da busca condiz com o encontrado por E.Carlsson (2013) onde é relatado a dificuldade de encontrar ferramentas práticas de usar sem dificuldades para a compreensão de seu funcionamento e que ainda estejam disponíveis ao público e atuais.

Entre os trabalhos encontrados selecionamos alguns que satisfazem alguns pontos do problema que definimos na seção anterior. Sourcerer (S. Bajracharya et al., 2014), GHTorent (G. Gousios, 2013) e Rminer (W. Shahbazian, 2010).

4.1 Sourcerer

Conforme S. Bajracharya et al. (2014), Sourcerer é uma infraestrutura para coleta, análise e recuperação de código aberto em larga escala que providencia uma fundação a partir da qual podem ser construídas ferramentas de ponta para busca, mineração e análise de código livre de projetos desenvolvidos em JAVA. Composta por diferenças ferramentas como:

- Um *crawler* baseado em plugins, já contando com plugins para Sourceforge, Java.net, Tigris, Google Code e Apache.
- Um criador de repositórios que gera repositórios no modelo criado para o Sourcerer e capaz de copiar os conteúdos de repositórios CVS ou SVN para preencher seus dados.
- Um gestor de repositórios que é utilizado para organização dos repositórios e otimização da capacidade do extrator de características.
- Um extrator de características dos códigos, capaz de extrair informações sobre a estrutura dos arquivos armazenados.
- Um importador de banco de dados capaz de armazenar as informações do extrator nos bancos de dados da infraestrutura Sourcerer.
- Um indexador de código para a indexação do conteúdo do repositório e permitindo a capacidade de busca textual de códigos.

Somadas estas ferramentas tornam o Sourcerer uma solução ponta a ponta para coleta, análise e recuperação de código aberto em larga escala. Facilitando a criação, o acesso e a manutenção de acervos de repositórios.

4.2 GHTorent

Conforme G. Gousios (2013), o projeto GHTorent tem como propósito criar uma cópia offline escalável e consultável dos dados do GitHub acessíveis através da API do mesmo. Devido ao limite de requisições da API do GitHub o projeto decidiu distribuir a coleta dos dados e assim aumentar a capacidade de informação sendo acessada ao mesmo tempo.

Em sua *homepage* consta que GHTorent já possui aproximadamente 6,5TB de dados brutos em JSON e 600.000.000 de linhas de metadados em um banco de dados MySQL com mais de 15 tipos diferentes de dados. O projeto foi capaz de obter grande parte da atividade no GitHub entre os anos 2012 e 2015 além de buscar adquirir dados de projetos importantes que precedem este intervalo.

4.3 Rminer

Conforme W. Shahbazian (2010), Rminer é uma ferramenta de extração e análise de códigos que utilizam os controladores de versão SVN, CVS ou Git. Ela foi implementada para a verificação da possibilidade de um modelo integrado para a mineração de repositórios de VCS distintos. Com isso ele determina dados em comum presentes nos três e obtém um nível de abstração para minerar os repositórios, porém procurando preservar as diferenças com tratamentos específicos para cada VC quando necessário.

Com o Rminer foi constatado a existência de conceitos em comum entre os controladores porém nem sempre há uma garantia que estes são realmente utilizados da mesma forma e conclui-se que um modelo integrado entre os três controladores de versão não é uma solução definitiva, mas é possível traçar pontos em comum entre eles e atingir uma camada de abstração desde que sejam observadas as suas diferenças.

4.4 Análise Comparativa

Cada um dos trabalhos apresentados nas seções anteriores deste capítulo tocam em, pelo menos, um ponto do problema definido na seção 3.2.

O Sourcerer apesar de ser uma ferramenta ambiciosa com um escopo maior do que o problema que queremos resolver. Ele foi desenvolvido com o intuito de armazenar repositórios criados ou clonados de um repositório que utiliza um controlador de versão. Apesar de apresentar suporte à CVS, SVN e ClearCase não há suporte para Git, e está limitada a projetos em JAVA. Para nosso objetivo de auxiliar a coleta de projetos a clonagem de repositórios é desnecessária.

O GHTorent é um acervo para projetos GitHub. O fato dele disponibilizar um banco de dados para acesso offline da enorme quantidade de dados acessíveis através da API do GitHub facilita o início de trabalhos que caso contrário estariam sujeitos aos limites impostos pela API. Ele não clona o repositório para obter seus dados porém não suporta projetos que não utilizam o GitHub, o que descarta projetos SVN e até mesmo projetos Git hospedados em outro local.

O Rminer verificou a possibilidade de integrar dados de projetos que usam VCS e conseguiu atingir níveis de abstração diferentes ao identificar conceitos em comum entre eles. Apresentando um foco na mineração e análise de códigos e seu desenvolvimento ele não dispõe de um mecanismo para buscar projetos online.

A API que propomos neste trabalho suporta projetos Git e SVN independente da linguagem de programação usada nestes. Ela é capaz de encontrar projetos Git hospedados no GitHub e, através do OpenHub, projetos SVN hospedados no Google Code. Visando muito menos informação a ser obtida e evitando clonagem de repositórios é observado que nossa API é inferior aos outros três projetos ao se comparar a diversidade de dados que podem ser obtidos. Apesar de apresentar restrições (explicadas no capítulo 4) de onde estão hospedados e nos tipos de VCS que os projetos buscados podem usar a API que propomos suporta todos os projetos Git ou SVN. A seguir está a tabela 4.1 com as diferenças mencionadas:

Tabela 4.1 – Comparação entre os trabalhos analisados

	VCS suportados	Capacidade de busca	Diversidade dos dados	Necessário clonagem
Sourcerer	CVS, SVN, ClearCase	Sourceforge, Java.net, Tigris, Google Code e Apache	Alta	Sim
GHTorent	Git*	GitHub	Alta	Não
Rminer	CVS, SVN e Git	Não	Alta	Sim
API proposta	Git e SVN	GitHub e OpenHub	Pouca	Não

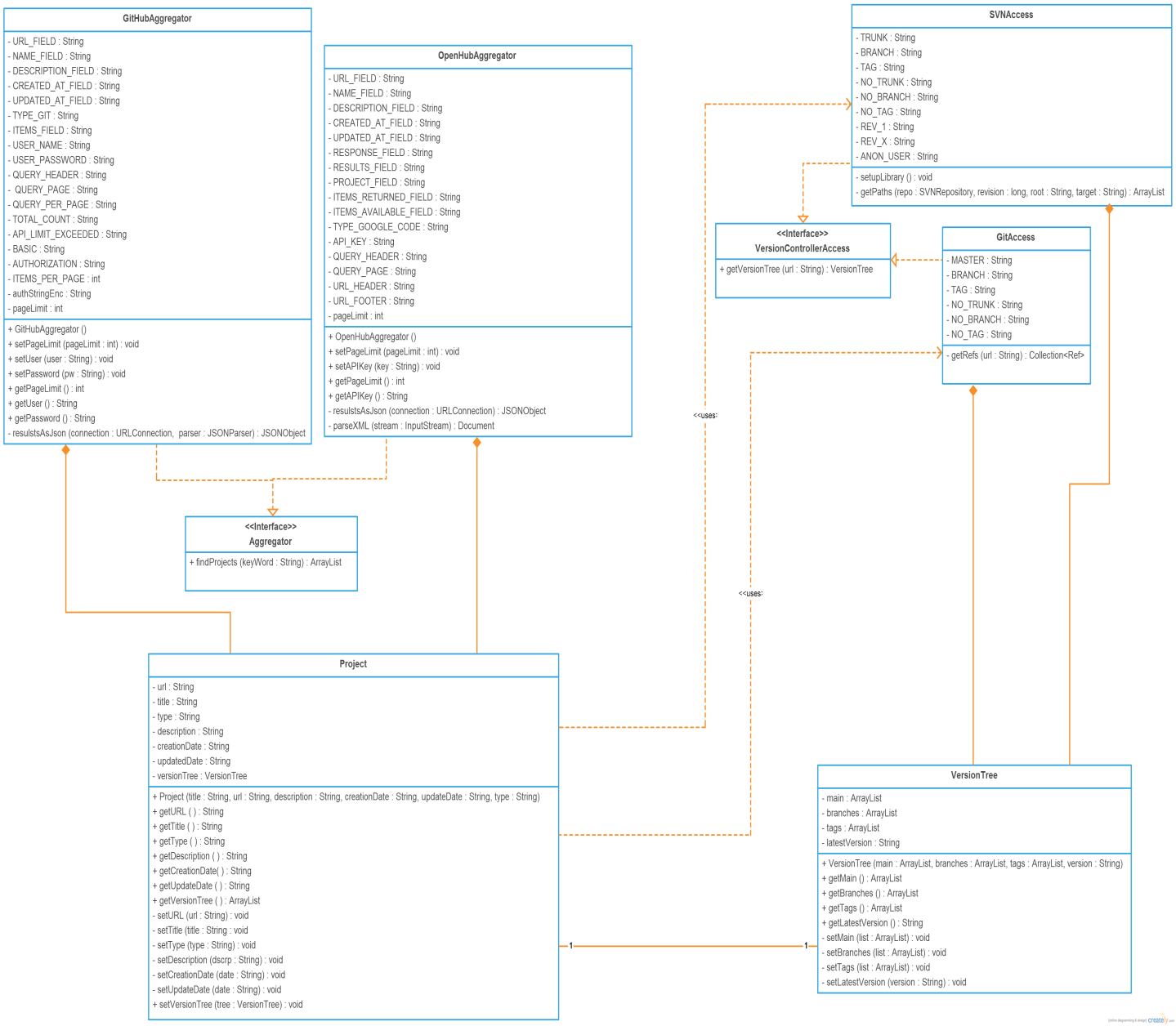
*Apenas projetos hospedados no GitHub

Fonte: Elaborado pelo Autor

5 A PROPOSTA

Neste capítulo falaremos sobre os detalhes da API proposta na seção 3.2 e tentaremos explicar as decisões tomadas no desenvolvimento da mesma. A API foi desenvolvida em Java Abaixo a figura 5.1 ilustra como ela foi estruturada.

Figura 5.1 – Diagrama UML completo da API



Fonte: Elaborado pelo Autor

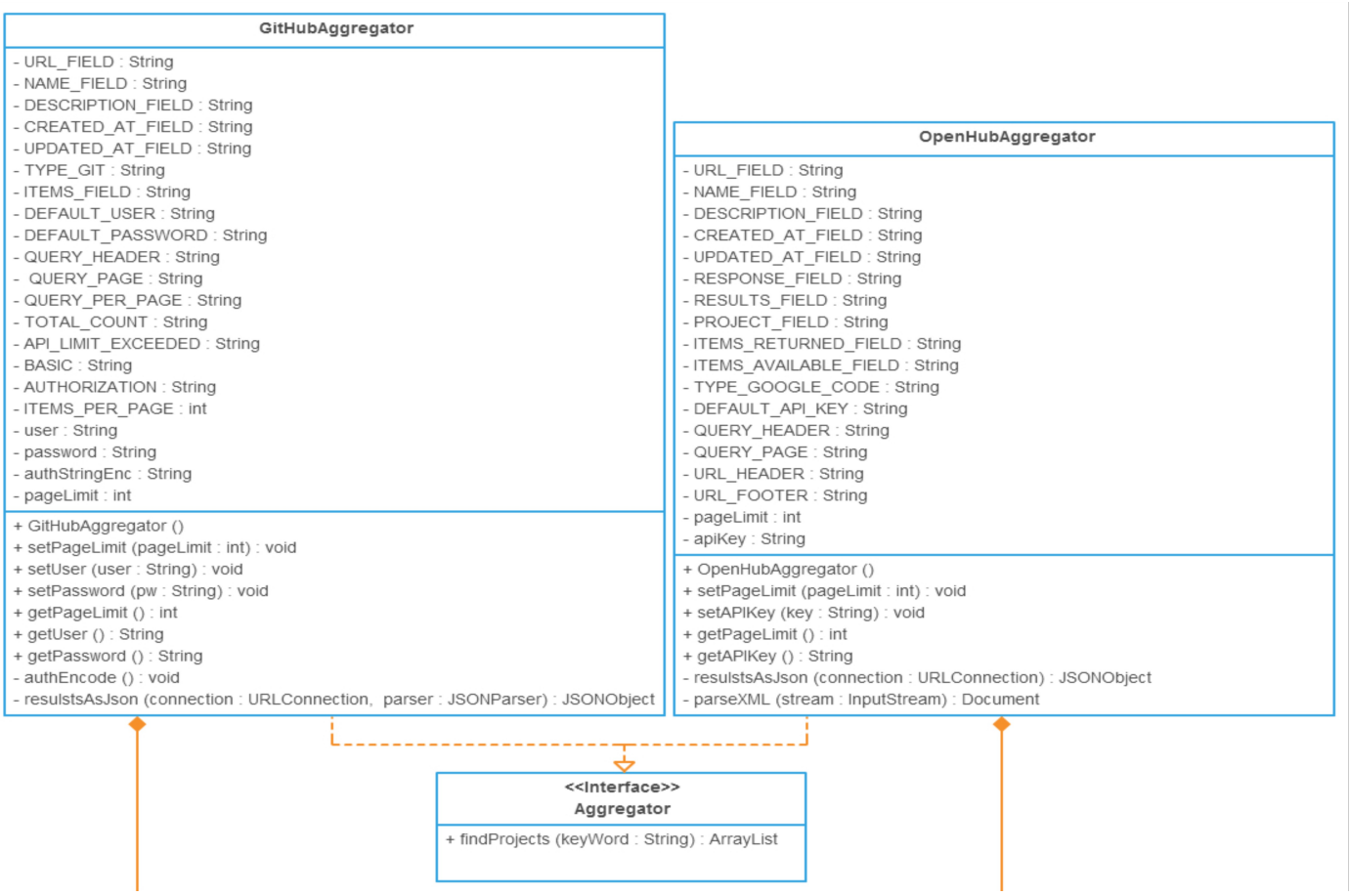
Iremos discutir o diagrama em três partes. A primeira versa sobre a busca dos projetos e contém a interface Aggregator, as duas classes, GitHubAggregator e OpenHubAggregator

provenientes desta interface. A segunda tratará sobre a obtenção da informação do versionamento, composta pela interface VersionControllerAccess e as duas classes que a implementam, GitAccess e SVNAccess. A última detalhará melhor as classes Project e VersionTree, e apresentará uma visão geral do todo.

Nas seções seguintes seguem explicaremos cada uma delas separadamente. Parte da proposta é o desacoplamento das classes de modo que qualquer dependência nas implementações possa ser substituída.

5.1 Busca de Projetos

Figura 5.2 – Diagrama UML da API ampliado no canto superior esquerdo



Fonte: Elaborado pelo Autor

Como pode ser visto na interface Aggregator, presente na figura 5.2, o propósito das classes eu a implementam é encontrar projetos. Como foi mencionado na seção 2.2 já existem ferramentas que facilitam nesta tarefa. Utilizamos a capacidade de busca da API do GitHub e da API do OpenHub nas classes GitHub Aggregator e OpenHubAggregator respectivamente.

5.1.1 GitHubAggregator

Para a implementação da classe *GitHubAggregator* trabalhamos com a API do GitHub. Apesar do nosso interesse em utilizá-la para a busca de projetos ela é, conforme descrito em seu site, otimizada para ajudar a encontrar itens específicos (e.g. um usuário específico, um arquivo específico em um repositório, etc.). Uma analogia à uma consulta no Google pode ser traçada em relação ao seu funcionamento. A busca conta também com alguns filtros como a linguagem de programação ou a data de criação do repositório. Para maiores detalhes sobre os filtros recomendamos a leitura da página da API referente à buscas. Os resultados de uma busca com filtro padrão são projetos que contém a palavra-chave em seus títulos ou descrições. A busca com GitHub implementada pela API proposta utiliza a função da API do GitHub de ordenar os resultados pela sua popularidade de forma descendente com o objetivo de facilitar a coleta de projetos que recebem mais atenção pela comunidade do GitHub.

Esta API é capaz de prover 100 resultados por página e até 1000 resultados em cada busca, porém ela possui um limite de requisições. Sem autenticação o limite é de 10 requisições por minuto, com autenticação esse limite é elevado para 30 requisições por minuto por usuário. Nossa implementação necessita 1 requisição inicial para cada busca seguida de 1 requisição por página de resultados. Obedecendo as restrições mencionadas anteriormente é possível encontrar até 900 projetos por minuto.

O acesso à API do GitHub é feito por requisições HTTP GET. Entre as opções de autenticação disponíveis julgamos suficiente a autenticação básica. Esta requer uma conta GitHub padrão e usa o login e senha codificados na autorização das requisições. O resultado de uma busca é devolvido em JSON e a partir das informações contidas nele instanciamos um *Project* para cada projeto. Segue abaixo uma breve explicação do conteúdo da classe *GitHubAggregator* conforme apresentado no diagrama acima:

- ITEMS_FIELD, TOTAL_COUNT: Constantes para obtenção do conteúdo nos campos contidos no JSON devolvido pela busca.
- URL_FIELD, NAME_FIELD, DESCRIPTION_FIELD, CREATED_AT_FIELD, UPDATED_AT_FIELD: Constantes para obter as informações referentes a um *Project* contidas nos campos do JSON devolvido pela busca. Apesar destas constantes estarem presentes em ambas as classes de busca, o texto contido nestas constantes não é o mesmo.

- QUERY_HEADER, QUERY_PAGE, QUERY_PER_PAGE, ITEMS_PER_PAGE: Constantes utilizadas na montagem do endereço HTTP para realizar a requisição de busca com a API do GitHub.
- BASIC, AUTHORIZATION: Constantes utilizadas na definição do tipo de autorização.
- TYPE_GIT: Constante para indicar o controlador de versão do projeto.
- use, password, authStringEnc: Variáveis utilizadas na construção da autenticação.
- pageLimit: Caso seja desejável limitar o número de páginas do resultado da busca contendo projetos a partir dos quais são criados *Projects*.
- GithubAggregator: Construtor onde user, password, authStringEnc e pageLimit recebem valores padrões em caso de novos valores não serem definidos.
- authEncode: Método que realiza a codificação da autenticação quando necessário.
- resultsAsJson: Método que extrai apenas as partes do JSON devolvido pela busca que representam os projetos encontrados.
- setPageLimit, setUser, setPassword: Métodos para definir os respectivos parâmetros.
- getPageLimit, getUser, getPassword: Métodos para obter os respectivos parâmetros.
- findProjects: Método da interface *Aggregator* implementado por esta classe para obter um conjunto de *Project* utilizando a busca de projetos a qual essa seção se refere. A implementação deste método nesta classe é específica para a busca de projetos Git hospedados no GitHub através da API do GitHub.

5.1.2 OpenHubAggregator

De maneira semelhante à API do GitHub a API do OpenHub permite realizar buscas por projetos. A principal diferença é que o OpenHub não apresenta apenas repositórios hospedados pelos serviços do GitHub, podendo assim também encontrar projetos que utilizam outros VCS como SVN e Mercurial em vez de apenas Git. Ele também mantém tags relevantes ao assunto do projeto, permitindo que projetos sejam encontrados mesmo sem conterem a palavra-chave em seu título ou descrição. A busca com OpenHub implementada pela API proposta utiliza a função da API do OpenHub de ordenar os resultados pelo grau de

atividade de forma descendente com o objetivo de facilitar a coleta de projetos que recebem mais atenção pela comunidade do OpenHub.

Porém a API do OpenHub é mais restritiva em outros quesitos. Para obter acesso à API é necessário uma *API Key* a ser usada para autenticação. Disponibilizando até 10 projetos por página sem um limite de resultados por busca, porém há um limite de 1000 requisições por dia por *API Key*. Em nossa implementação novamente necessitamos 1 requisição inicial por busca seguida de 1 requisição por página. Porém a informação referente ao tipo de sistema de controle de versão que um projeto usa e a localização do seu repositório não é disponibilizada sem o uso de uma requisição extra em cada projeto para mais detalhes sobre o mesmo. Com isso a capacidade é de encontrar até 900 projetos por dia.

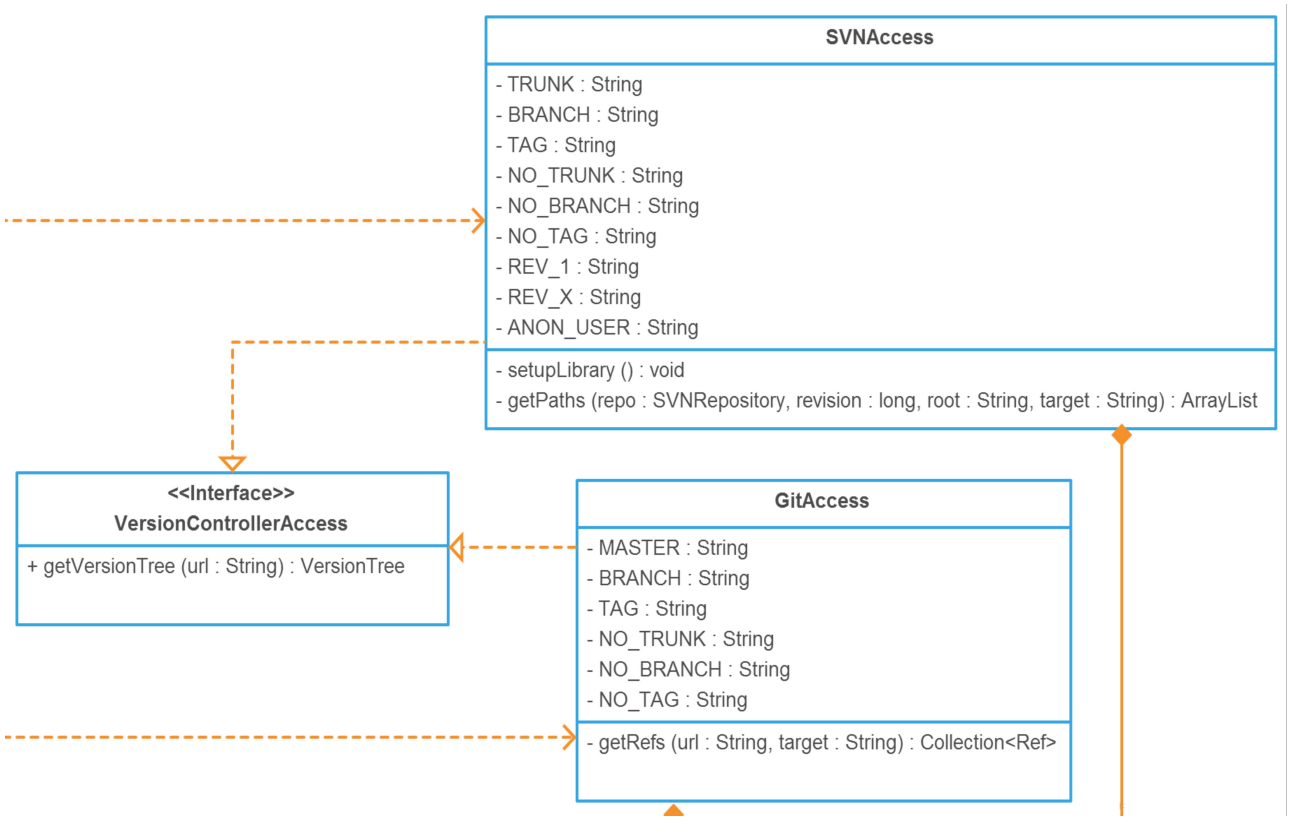
Assim como com a API do GitHub o acesso à API do OpenHub é através de requisições HTTP GET. A autenticação é feita passando a *API Key* como um parâmetro. O resultado de uma busca é devolvido em XML, convertido para JSON e com as informações contidas nele instanciamos um *Project* para cada projeto. Segue abaixo uma breve explicação do conteúdo da classe *OpenHubAggregator* conforme apresentado no diagrama acima:

- RESPONSE_FIELD, RESULTS_FIELD, PROJECTS_FIELD, ITEMS_RETURNED_FIELD, ITEMS_AVAILABLE_FIELD: Constantes para obtenção do conteúdo nos campos contidos no JSON devolvido pela busca.
- URL_FIELD, NAME_FIELD, DESCRIPTION_FIELD, CREATED_AT_FIELD, UPDATED_AT_FIELD: Constantes para obter as informações referentes a um *Project* contidas nos campos do JSON devolvido pela busca. Apesar destas constantes estarem presentes em ambas as classes de busca, o texto contido nestas constantes não é o mesmo.
- QUERY_HEADER, QUERY_PAGE: Constantes utilizadas na montagem do endereço HTTP para realizar a requisição de busca com a API do OpenHub.
- URL_HEADER, URL_FOOTER: Constantes utilizadas para montar a URL do repositório do projeto hospedado pelo serviço Google Code.
- TYPE_GOOGLE_CODE: Constante para indicar o controlador de versão do projeto.
- apiKey: Variável contendo a *API Key* que será utilizada para autenticação.
- pageLimit: Caso seja desejável limitar o número de páginas do resultado da busca contendo projetos a partir dos quais são criados *Projects*.

- OpenhubAggregator: Construtor onde apiKey e pageLimit recebem valores padrões em caso de novos valores não serem definidos.
- parseXML: Método que auxilia na leitura do resultado XML para a conversão em JSON.
- resultsAsJson: Método que converte o XML obtido da busca em um JSON contendo as informações dos projetos encontrados pela busca.
- setPageLimit, setAPIKey: Métodos para definir os respectivos parâmetros.
- getPageLimit, getAPIKey: Métodos para obter os respectivos parâmetros.
- findProjects: Método da interface *Aggregator* implementado por esta classe para obter um conjunto de *Project* utilizando a busca de projetos a qual essa seção se refere. A implementação deste método nesta classe é específica para a busca de projetos SVN hospedados no Google Code através da API do OpenHub.

5.2 Acesso ao Controlador de Versão

Figura 5.3 – Diagrama UML da API ampliado no canto superior direto



Fonte: Elaborado pelo Autor

A finalidade das classes GitAccess e SVNAccess, que implementam a interface VersionControllerAccess, é abrir uma conexão com seus respectivos controladores de versão (Git e SVN). Com o acesso estabelecido é possível obter a informação de versionamento visada do repositório.

Para a implementação do acesso aos controladores de versão foi constatada a existência de APIs para Java as quais foram criadas especificamente para trabalhar com Git e SVN, usamos então a API JGit com a classe GitAccess e a API SVNKit com a classe SVNAccess.

5.2.1 GitAccess

Devido à sua característica de ser um controlador de versão descentralizado, o Git é limitado ao que se refere à acesso ao repositório sem uma cópia local. Visto que nosso objetivo é apenas extrair algumas informações visamos evitar a criação de tal cópia. Por mais limitados que sejam entre os comandos que o Git dispõe existia um com a capacidade de obter a lista de referências a *branches* e *tags* que procurávamos.

Com o auxílio da API JGit acessamos o repositório Git alvo com uma chamada remota a partir de seu URL. O retorno dela é a lista de referências existentes das *tags* e *branches* do repositório. A partir dessas informações é possível então criar a *VersionTree* do projeto correspondente ao URL do repositório Git acessado. Segue abaixo uma breve explicação do conteúdo da classe *OpenHubAggregator* conforme apresentado no diagrama acima:

- MASTER, BRANCH, TAG: Constantes utilizadas para indicar o tipo de referência a ser obtido.
- NO_MASTER, NO_BRANCH, NO_TAG: Constantes utilizadas quando não foi possível encontrar referências de um respectivo tipo.
- getRefs: Método que acessa o repositório Git remotamente e obtém as referências do tipo de informação especificado (master, branch ou tag).
- getVersionTree: Método da interface *VersionControllerAccess* implementado por esta classe para obter a *VersionTree* de um repositório Git correspondente ao URL recebido. Com as referências obtidas pelo método getRefs constrói-se uma *VersionTree*.

5.2.2 SVNAccess

Ao contrário do Git, o controlador de versão SVN apresenta maior liberdade para trabalhar com um repositório sem uma cópia local. Uma das vantagens é o fato que a estrutura de um diretório SVN pode ser acessada por URL desde que o mesmo esteja online. Por consequência obtemos não apenas a lista com *trunk, branches* e *tags* como também as URL de seus respectivos diretórios.

Utilizando a da API SVNKit obtemos acesso ao repositório SVN alvo a partir de seu URL. Com isso obtemos a revisão mais recente e os caminhos para *trunk, tags* e *branches* do repositório. A partir dessas informações é possível então criar a *VersionTree* do projeto correspondente ao URL do repositório SVN acessado. Segue abaixo uma breve explicação do conteúdo da classe *OpenHubAggregator* conforme apresentado no diagrama acima:

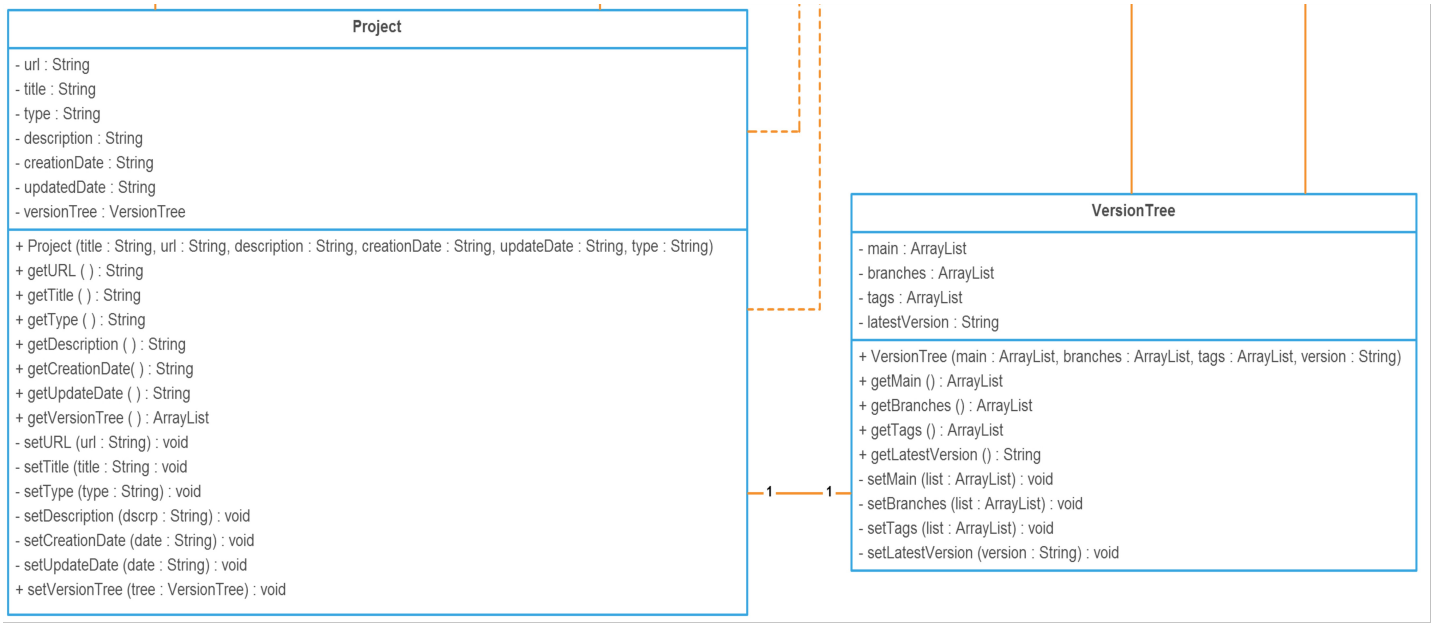
- TRUNK, BRANCH, TAG: Constantes utilizadas para indicar o tipo de caminho a ser obtido.
- NO_TRUNK, NO_BRANCH, NO_TAG: Constantes utilizadas quando não foi possível encontrar caminhos de um respectivo tipo.
- setupLibrary: Inicialização da API SVNKit para acesso SVN.
- getPaths: Método que acessa o repositório SVN e obtém os caminhos do tipo de informação especificado (trunk, branch ou tag).
- getVersionTree: Método da interface *VersionControllerAccess* implementado por esta classe para obter a *VersionTree* de um repositório SVN correspondente ao URL recebido. Com os caminhos referências obtidos pelo método getPaths constrói-se uma *VersionTree*.

5.3 Project, VersionTree e a interação do todo

5.3.1 Project e VersionTree

As classes *Project* e *VersionTree* são os dados criados e manuseados pelas classes apresentadas anteriormente. Um *Project* contém uma *VersionTree* conforme o diagrama na figura 5.4. Os atributos presentes nas classes são os dados que nos interessam extrair.

Figura 5.4 – Diagrama UML da API ampliado na metade inferior



Fonte: Elaborado pelo Autor

Em *Project* temos:

- *URL*: o endereço online de onde se encontra o repositório.
- *Title* o nome do repositório.
- *Type:* É um identificador do controlador de versão que o projeto usa.
- *Description*: quando existente, uma descrição textual do repositório
- *Creation Date:* Data em que o projeto foi criado.
- *Update Date:* Data em que o projeto foi atualizado por último.
- *VersionTree*: quando existente, os endereços de onde as versões passadas do repositório podem ser obtidas.

E em *VersionTree* temos:

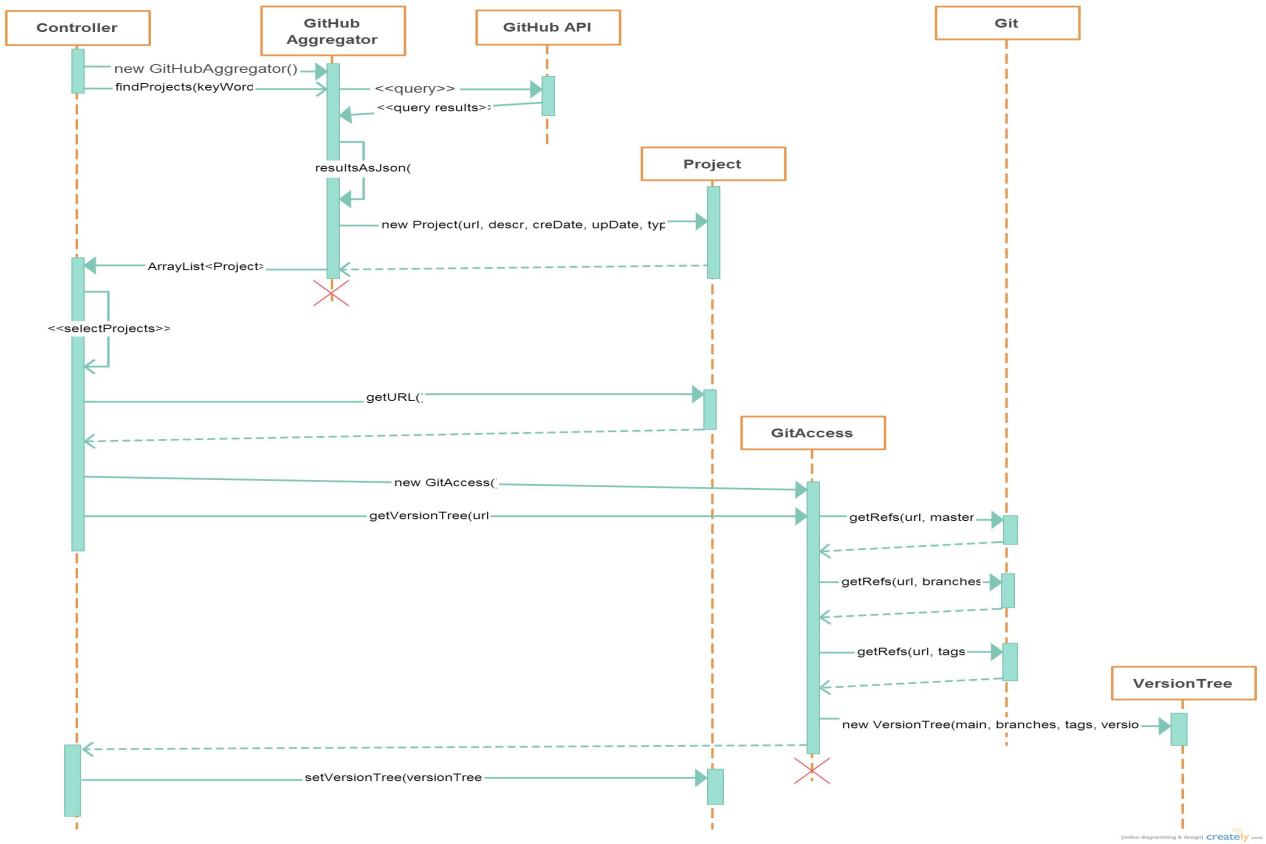
- *Main:* É o corpo principal do repositório. *Trunk* no SVN e *Master* no Git.
- *Branches:* São criados para avançar o projeto sem afetar o corpo principal.

- *Tags:* Quando o projeto atinge um certo marco o usuário pode criar uma tag para salvar uma referência ao estado do repositório neste ponto a qual pode ser retomada posteriormente.
- *LatestVersion:* No SVN é a *revision* mais recente. No Git devido ao modelo descentralizado optamos por usar o nome da *tag* mais recente.

5.3.2 Funcionamento da API

A figura 5.5 a seguir ilustra o funcionamento esperado da API com um diagrama de sequência. O diagrama está focando no processo envolvido ao realizar uma busca utilizando apenas o GitHub e projetos que usam Git para tentar evitar que a compreensão seja dificultada. O processo usando OpenHub e projetos SVN ocorre de maneira similar e na explicação que segue mencionaremos as poucas diferenças. No diagrama foi introduzida uma classe *Controller* para fazer uso das classes da API e auxiliar na troca de informação entre elas. De certa forma ele faz o papel de uma interface de usuário.

Figura 5.5 – Diagrama de Sequência



Fonte: Elaborado pelo Autor

O conjunto da nossa API interage como um todo para criar um ou mais *Project* com seus atributos preenchidos. Um *GitHubAggregator* é instanciado para fazer a conexão com a API de busca do GitHub. Invocamos o método *findProjects(keyword)* fornecendo a palavra-chave que é então utilizada como parâmetro da *query*. O resultado da busca com a API do GitHub são dados brutos em JSON (com a API do OpenHub estão em *xml*). Com a invocação do método *resultsAsJson()* colocamos apenas a seção destes dados que contém as informações dos projetos em um objeto JSON. Entre essas informações estão os parâmetros para a invocação do construtor de cada *Project.* O *Controller* então recebe um *ArrayList* de *Project* os quais já possuem seus atributos preenchidos com exceção da *VersionTree*.

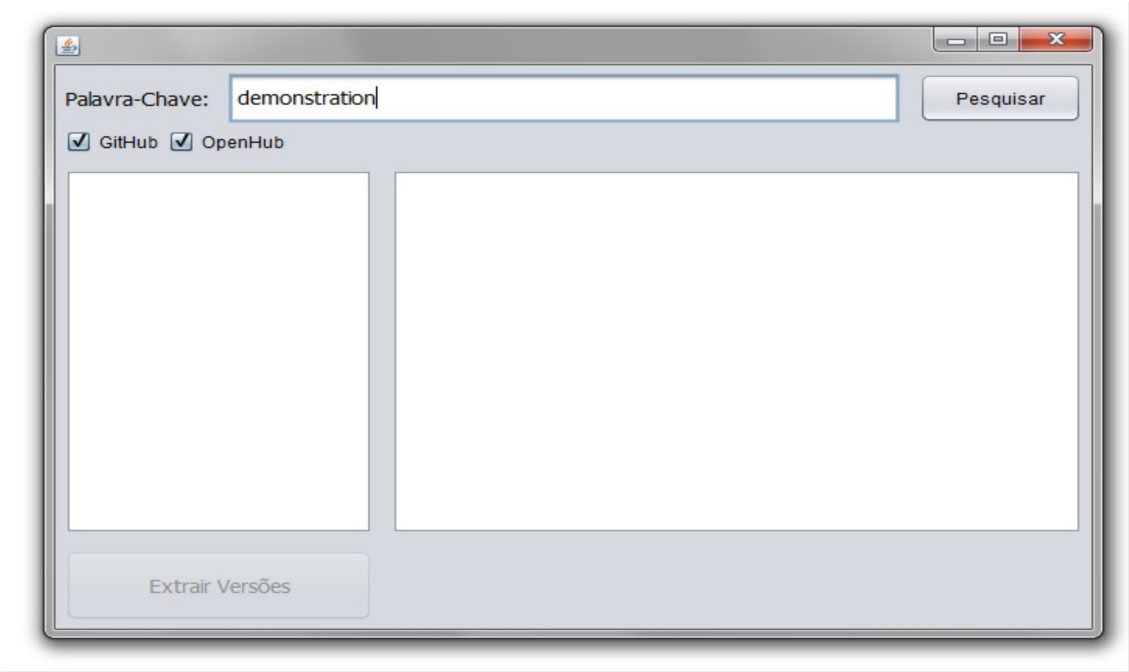
Na sequência é feita a escolha de quais os objetos *Project* criados irão continuar para receber as suas *VersionTree.* Para criarmos uma *VersionTree* é necessário o acesso ao controlador de versão adequado ao tipo do projeto. Um projeto Git portanto precisa de uma instância de *GitAccess*. O *Controller* acessa o URL de um *Project* e chama o método *getVersionTree(url)* do *GitAccess*. Este acessa o repositório Git remotamente e obtém as referências que irão compor os parâmetros *main, branches* e *tags* do construtor da *VersionTree*. Para Git o parâmetro *latestVersion* é aproximado a partir de suas *tags* enquanto que no SVN ele é obtido diretamente. Com a *VerstionTree* pronta ela é devolvida ao *Controller* e este completa o *Project* correspondente com o método *setVersionTree(versionTree)*.

Com esta estrutura a API desacopla a busca por projetos do acesso ao controlador de versões. Desde que haja um elemento que instancie *Project* e um elemento que atribua uma *VersionTree* para este, obtemos o *Project* completo.

5.3.3 Ferramenta que utiliza a API

Para demonstrar o uso da API também desenvolvemos uma ferramenta que integra seus componentes fazendo o papel do *Controller* mencionado na seção anterior. A seguir estão imagens ilustrando o uso da ferramenta:

Figura 5.6 – Captura de tela da demonstração da ferramenta

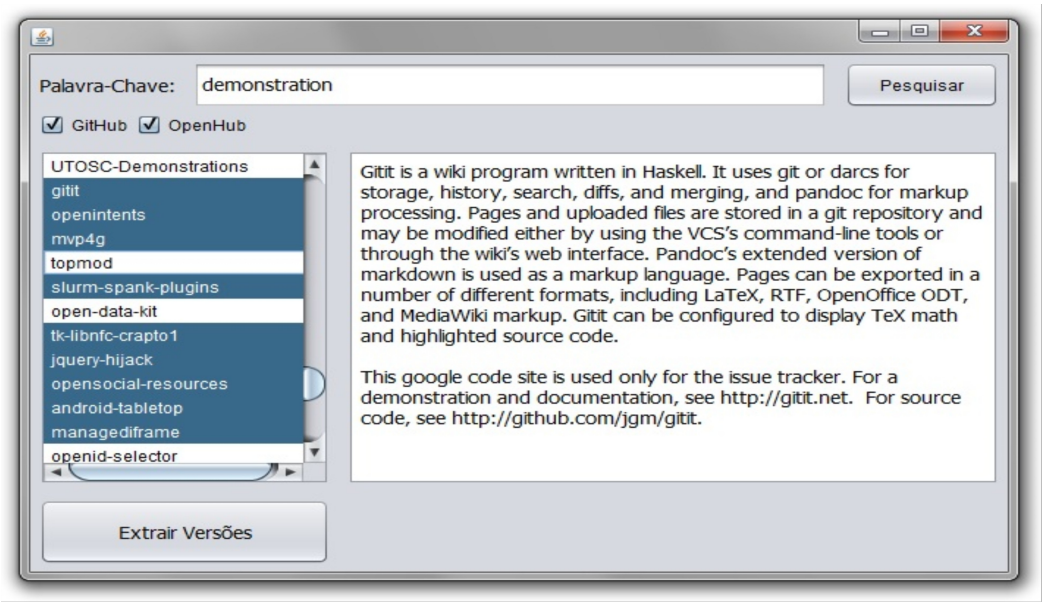


Fonte: Elaborado pelo Autor

A figura 5.6 exemplifica a entrada da palavra-chave para o início da busca por projetos. Nota-se a possibilidade de escolher se serão inclusos resultados do GitHub e/ou OpenHub. Para prosseguir basta pressionar o botão “Pesquisar”.

A figura 5.7 abaixo apresenta então os resultados da busca. Cada item na lista à esquerda corresponde a um *Project* com todos seus atributos exceto a *VersionTree*. À direita é disponibilizada a descrição do projeto para auxiliar na decisão de quais projetos permanecerão na lista. Para prosseguir pressiona-se o botão “Extrair Versões”.

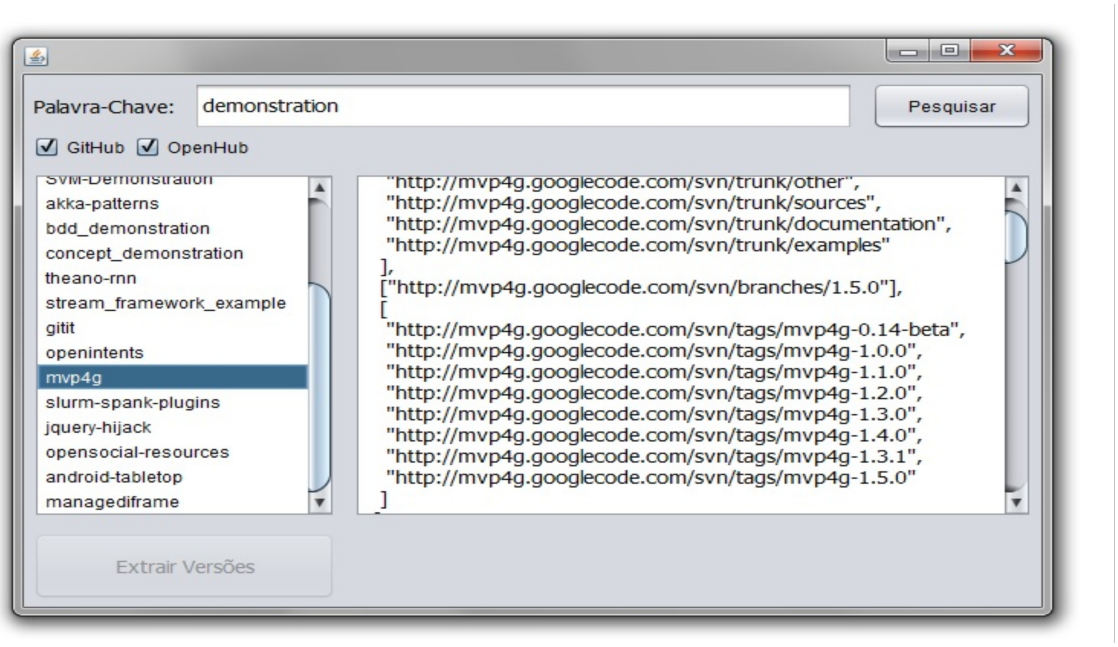
Figura 5.7 – Captura de tela da demonstração da ferramenta



Fonte: Elaborado pelo Autor

Na figura 5.8 na próxima página vemos então, à esquerda, a lista de projetos selecionados no passo anterior, cujos *Projects* agora estão completos após a adição das suas respectivas *VersionTrees*. À direita ficam exibidos todos os atributos do *Project* selecionado. Um arquivo de texto é criado com todos os *Project*s da lista em formato JSON.

Figura 5.8 – Captura de tela da demonstração da ferramenta



Fonte: Elaborado pelo Autor

Após o desenvolvimento da ferramenta foram realizados alguns testes para avaliação da mesma. Os testes são constituídos pela realização de buscas de projetos importantes que utilizam Git ou SVN seguidos a extração dos metadados e das listas de branches e tags desses. Nos resultados são apresentados os projetos encontrados com cada par de palavra-chave e

mecanismo de busca, o tempo médio das buscas com o GitHub e das buscas com o OpenHub, e os tempos para a extração das listas de branches e tags de cada um dos projetos. Cinco projetos importantes foram selecionados, estes são: Hadoop, Spark, Cassandra, Neo4j e Elasticsearch. As palavras-chaves relevantes a eles que foram utilizadas para as buscas foram: “Hadoop”, “NoSQL”, “MapReduce”, “Search”, “Graph” e “Distributed Computing”. O número de resultados encontrados foi limitado à 100 projetos no GitHub e 100 projetos no OpenHub e apenas repositórios dos cinco projetos selecionados serão considerados apesar de existirem projetos que estão relacionados a esses que também estavam presentes entre todos os encontrados durante as buscas. Ressaltamos que o GitHub só encontra projetos hospedados no mesmo e que usam Git, e o OpenHub está limitado pela API proposta a encontrar apenas projetos que usam SVN ou Git. Abaixo seguem os resultados das buscas com o GitHub:

- Com a palavra-chave “Hadoop”, o GitHub encontrou um *mirror* do repositório do projeto Hadoop.
- Com a palavra-chave “Search”, o GitHub encontrou o repositório do projeto Elasticsearch.
- Com a palavra-chave “Graph”, o GitHub encontrou o repositório do projeto Neo4j
- Com as palavras-chave “NoSQL”, “MapReduce” e “Distributed Computing”, o GitHub encontrou nenhum dos cinco projetos.
- A média do tempo levado pela API proposta para obter os metadados de 100 projetos encontrados com a API do GitHub foi de aproximadamente 3,68 segundos.

Abaixo seguem os resultados das buscas com o OpenHub:

- Com as palavras-chave “Hadoop”, “MapReduce” e “Distributed Computing”, o OpenHub encontrou os repositórios dos projetos Hadoop e Spark.
- Com a palavra-chave “NoSQL”, o OpenHub encontrou o repositório do projeto Cassandra.
- Com a palavra-chave “Search”, o OpenHub encontrou o repositório do projeto Elasticsearch.
- Com a palavra-chave “Graph”, o OpenHub encontrou o repositório do projeto Spark. Ele também encontrou o projeto Neo4j porém esse apresentava a informação de diversos repositórios relacionados a ele impedindo a obtenção automática do seu repositório principal.
- A média do tempo levado pela API proposta para obter os metadados de 100 projetos encontrados com a API do OpenHub foi de aproximadamente 46,78 segundos.

Com estes resultados observamos que a obtenção dos metadados com a API do GitHub é mais rápida. Isso pode ser consequência da transformação de XML para JSON que a API proposta realiza sobre os dados obtidos com a API do OpenHub ou também devido à necessidade com essa de realizar uma requisição extra para cada projeto. Também é possível observar que a API do OpenHub obteve mais êxito para encontrar os cinco projetos selecionados com as palavras-chave selecionadas. Isso pode ser consequência do fato que a API do OpenHub utiliza tags relevantes ao conteúdo dos projetos além de também buscar a palavra-chave no título e descrição desses ou também e também que essa não está limitada a buscar apenas por projetos hospedados no GitHub.

Com essas buscas foi possível encontrar os repositórios dos cinco projetos selecionados. O teste continua com a criação da VersionTree desses a partir do acesso aos seus respectivos repositórios. Todos os cinco projetos selecionados utilizam o sistema de controle de versão Git. Entre eles o que mais demorou para a criação da VersionTree foi o projeto Spark com um tempo aproximado de 6,24 segundos. O repositório do projeto Spark contém 12 branches e 35 tags. Enquanto que o repositório do projeto Hadoop, cuja criação da VersionTree foi a segunda mais rápida e demorou aproximadamente 2,80 segundos, conta com o maior número de branches e tags, 142 e 202 respectivamente. O tamanho da amostra é pequeno e com ela não foi possível estabelecer uma relação direta entre o tempo que demora para a criação da VersionTree e a quantidade de branches e tags. Visto que nenhum dos cinco projetos selecionados utiliza o sistema de controle SVN foi realizada a criação da VersionTree do repositório SVN do projeto Subversion para observar o tempo que esta leva. Contando com 90 branches e 217 tags a operação foi concluída em aproximadamente 6,03 segundos.

6 CONCLUSÃO

Com este texto falamos sobre o potencial que a coleta de informações contidas em repositórios existentes e as dificuldades que a variedade existente na representação destes apresenta. Com o capítulo 5 apresentamos a API proposta, que realiza o processo de encontrar repositórios a partir de um critério de busca e coletar a lista de *branches* e *tags* contidas neles sendo então possível utilizar tais dados posteriormente e desta forma auxiliar em pesquisas.

A API foi implementada com suporte para busca nos acervos do GitHub e do OpenHub, e capacidade de extração nos VCS Git e SVN. Ela também possibilita a capacidade de adicionar suporte a outros acervos e outros VCS além dos já suportados. Não havendo dependência direta entre a parte de busca e a parte de acesso ao repositório a extensão da API pode ser feita em apenas uma delas caso desejado.

Como continuação do trabalho a adição de suporte de busca a outros acervos é importante visto que os suportados apresentam limitações que forçam a restrição da variedade dos VCS que os projetos encontrados usam. Também podem ser adicionadas a utilização de outros filtros de busca e ordenação de resultados para melhorar os mecanismos de busca. A estrutura da API pode ser revista para o uso de *patterns* com o objetivo de permitir melhor flexibilidade no desenvolvimento das adições mencionadas.

REFERÊNCIAS

BACHMANN, A.; BERNSTEIN, A. Software Process Data Quality and Characteristics. In: **IWPSE-Evol '09**, Amsterdam, Netherlands, 2009.

ZIMMERMANN, T.; ZELLER, A.; WEISSGERBER, P.; DIEHL, S. Mining version histories to guide software changes. In: **IEEE Transactions on Software Engineering**, 31(no.6) (2005): 429-445.

CHACON, S.; STRAUB, B. **Pro git**. Apress, 2009.

LOELIGER, J.; MCCULLOUGH, M. **Version Control with Git: Powerful tools and techniques for collaborative software development**. O'Reilly Media, Inc., 2012. APA

Collins-Sussman, B.; FITZPATRICK, B.; PILATO, M. **Version control with subversion**. O'Reilly Media, Inc., 2004.

BIRD, C.; RIGBY, P. C.; BARR, E. T.; HAMILTON, D. J.; GERMAN, D. M.; DEVANBU, Prem. The promises and perils of mining git. In: **Mining Software Repositories**, 2009. MSR'09. 6th IEEE International Working Conference on, pp. 1-10. IEEE, 2009.

SHAHBAZIAN, W. **Rminer: An integrated model for repository mining using rascal a feasibility study**. Diss. Master Thesis, 2010.

MCCABE, T. J. A Complexity Measure. In: **IEEE Transactions on Software Engineering**, Vols. SE-2, pp. 308-320, 1976.

BAJRACHARYA, S.; OSSHER, J.; LOPES, C. Sourcerer: An infrastructure for large-scale collection and analysis of open-source code. In: **Science of Computer Programming**, 79 (2014): 241-259.

GOUSIOS, G. The GHTorent dataset and tool suite. In: **Proceedings of the 10th Working Conference on Mining Software Repositories**, pp. 233-236. IEEE Press, 2013.

Website do GHTorent. Último acesso em Julho, 2015. http://ghtorrent.org/

Website da API de busca do GitHub. Último acesso em Julho, 2015. https://developer.github.com/v3/search/

Website da API de busca do OpenHub. Último acesso em Julho, 2015. https://github.com/blackducksw/ohloh_api

Website da API Jgit. Último acesso em Julho, 2015. https://eclipse.org/jgit/

Website da API SVNKit. Último acesso em Julho, 2015. http://svnkit.com/

Website da ferramenta utilizada para criar diagramas. Último acesso em Julho, 2015. https://creately.com/

Website do Hadoop. Último acesso em Julho, 2015. https://hadoop.apache.org/

Website do Cassandra. Último acesso em Julho, 2015. http://cassandra.apache.org/

Website do Spark. Último acesso em Julho, 2015. http://spark.apache.org/

Website do Neo4j. Último acesso em Julho, 2015. http://neo4j.com/

Website do Elasticsearch. Último acesso em Julho, 2015. https://www.elastic.co/products/elasticsearch