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

PAULA BURGUÊZ

Módulo Adapter para acesso a dados de sistemas de controle de versão

Trabalho de Graduação.

Prof. Dr. Marcelo Soares Pimenta
Orientador

Porto Alegre, dezembro de 2013.

“A source code control system is a giant UNDO key - a project-wide time machine.”
— ANDY HUNT AND DAVE THOMAS

AGRADECIMENTOS

Agradeço a minha família por todo o apoio ao longo dessa caminhada, desde a escolha do curso até a conclusão do mesmo. Aos meus pais, meu muito obrigada por serem os meus maiores exemplos e por me motivarem a seguir meus objetivos e sonhos, sem desistir na primeira dificuldade. A minha irmã caçula, muito obrigada pela amizade, por todo amor e incentivo em todos os anos da minha vida. A minha irmã mais velha e sua família, muito obrigada por todo o carinho e apoio, mesmo de longe. A minha madrinha, muito obrigada pela paciência e dedicação a mim.

Agradeço também ao Instituto de Informática da Universidade Federal do Rio Grande do Sul, pelo ensino de qualidade e por todo suporte ao longo desses anos, de professores e funcionários. Muito obrigada aos meus colegas pela força, pelos projetos desenvolvidos juntos e por todos os momentos compartilhados fora da universidade. Muito obrigada aos colegas e amigos que compartilharam comigo a experiência nos Estados Unidos, por todo carinho e apoio ao longo de 2012. Obrigada especialmente ao meu orientador Marcelo Pimenta e ao doutorando Fábio Petrillo, pela orientação, paciência e apoio nesse trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

VCS	Sistema de controle de versão
API	Application Programming Interface
SVN	Subversion
GUI	Graphic User Interface
URL	Uniform Resource Locator

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Sistemas de controle de versão são ferramentas largamente utilizadas no mundo inteiro, para auxiliar no desenvolvimento colaborativo e concorrente. São ferramentas que permitem que mais de um desenvolvedor trabalhe no mesmo arquivo, além de fornecer o histórico daquele arquivo ao longo das suas versões. Apesar de todos os sistemas serem baseados nos mesmos conceitos básicos, eles possuem terminologia e comandos diferentes. O objetivo deste trabalho é criar uma abstração para sistemas de controle de versão, gerando um modelo genérico para o desenvolvimento de uma API que permita a leitura dos dados do projeto independente do sistema. O objetivo principal dessa API é recuperar arquivos e dados sobre versionamento dos mais diversos projetos.

Palavras-Chave: Sistemas de Controle de Versão, Git, Subversion

Adapter Module for data access in version control systems

ABSTRACT

Version control systems are tools widely used around the worldwide, to support collaborative and concurrent development. They are tools that support more than one developer working on the same file, while also providing a history of that file along versions. Despite the fact that all systems are based on the same fundamental concepts, they have different terminology and commands. This work aims to create an abstraction for version control systems, bringing up a generic model for the development of an API that provides methods to read project data, regardless the system. The main goal of this API is to retrieve files and data versioning from several projects.

Keywords: Version Control System, Git, Subversion

1 INTRODUÇÃO

Sistemas de controle de versão (VCS) são ferramentas para desenvolvimento colaborativo (CHACON, 2009), utilizadas pela maioria das empresas de software (BALL et al. 1997). São sistemas que permitem o gerenciamento de versões de um software, não apenas do código-fonte, mas também de diversos outros tipos de arquivos. Além disso, são sistemas que mantém um histórico completo do desenvolvimento, como datas e autores das alterações dos arquivos, entre outros. Com isso, é possível recuperar versões anteriores do código-fonte (OTTE, 2008).

Outro aspecto importante é o suporte a desenvolvimento concorrente. A solução clássica para trabalhar com concorrência é o mecanismo de *lock-modify-unlock*, que bloqueia um recurso para ser utilizado por um processo e só libera quando o processo termina as modificações necessárias. Esse mecanismo é inviável para um VCS, visto que os usuários geralmente esquecem de desbloquear os recursos, além de permitir que apenas um usuário altere o arquivo. O mecanismo utilizado é o *copy-modify-merge*, que copia para um ou mais usuários o recurso para ser modificado, juntando as versões e detectando possíveis conflitos após as modificações (OTTE, 2008).

Por armazenar informações importantes do histórico de um software, um VCS é geralmente uma fonte básica para o campo de pesquisa conhecido como Análise de Evolução de Software. Pesquisadores dessa área buscam entender as mudanças no sistema, a fim de descobrir por que a manutenção torna-se mais difícil a cada modificação e de tentar ajudar a reduzir custos (CHERAIT et al, 2012).

Atualmente, existem diversos sistemas de controle de versão no mercado, seja centralizados como o SVN, ou distribuídos, como o Git. Apesar de serem fortemente baseados nas mesmas ideias, muitos desses sistemas possuem terminologia própria, diferentes comandos e interface, conforme observado durante o estudo desses sistemas para este trabalho. Isso pode trazer algumas dificuldades para a comparação e avaliação dos projetos, especialmente para pesquisadores de áreas que analisam dados e informações de versionamento desses projetos.

1.1 Motivação

A motivação para este trabalho surgiu de uma necessidade do projeto do doutorando Fábio Petrillo, que faz parte de um grupo da UFRGS que trabalha na área de Visualização de Software, uma subárea da Engenharia de Software. A área é responsável por auxiliar na compreensão de software desenvolvendo formas de representar visualmente as suas propriedades. Assim como outros pesquisadores da área, o grupo utiliza os dados de versionamento como fonte básica de pesquisa. Esses

projetos analisados geralmente utilizam diferentes sistemas de controle de versão, tornando a coleta desses dados dependente do sistema.

1.2 Objetivo

O objetivo deste trabalho é estudar alguns dos sistemas de controle de versão disponíveis e elaborar uma abstração dos seus conceitos, possibilitando a implementação de um módulo que permita a recuperação das informações de projetos de diferentes sistemas. O principal objetivo é elaborar uma API que permita a coleta de dados e de informações sobre versionamento.

Com este trabalho, busca-se oferecer suporte aos usuários de sistemas de controle de versão, especialmente a pesquisadores de áreas como Análise de Evolução de Software, Visualização de Software e Mineração de Dados. São áreas da computação que contribuem para melhorar o entendimento do processo de desenvolvimento de um software, baseando fortemente suas pesquisas em estatísticas e análises de dados de projetos.

A escolha dos sistemas de controle de versão para este trabalho foi baseada nos sistemas mais populares. Subversion e Git são dois dos sistemas mais utilizados para projetos Open Source atualmente (KLEINE, 2012). Foi escolhido pelo menos um sistema distribuído e um centralizado para que os principais conceitos e funcionalidades fossem analisadas e incluídas na API.

1.3 Organização do texto

Para cumprir os objetivos propostos, este trabalho foi dividido em 5 capítulos. Após a introdução, no capítulo 2 serão apresentados os VCS analisados, com uma breve explicação sobre as principais diferenças entre sistemas centralizados e distribuídos, seguida das características próprias desses sistemas.

No capítulo 3, será apresentada a solução buscada por este trabalho, com detalhes sobre o funcionamento da API proposta. A implementação básica dessa API é o foco do capítulo 4, bem como exemplos de uso e as APIs utilizadas para cada sistema. No capítulo 5, será apresentado um estudo de caso. Por fim, no capítulo 6, será apresentada a conclusão do trabalho.

2 SISTEMAS DE CONTROLE DE VERSÃO

Nesse capítulo, serão introduzidos os Sistemas de Controle de Versão analisados para este trabalho. Primeiro, serão apresentados os conceitos e as principais diferenças entre os VCS, focando nos distribuídos e centralizados. Por fim, uma análise dos VCS utilizados neste trabalho, Git e SVN, bem como suas características próprias.

2.1 Centralizados x Distribuídos

Os sistemas de controles centralizados trabalham em um repositório central, onde todos os arquivos, incluindo seus históricos, são armazenados. Cada estado de um arquivo é identificado pela versão, denominada *version* ou *revision*, que pode ser recuperada pelo usuário com uma operação de *checkout*. A cópia local do usuário, editável e sem histórico, é chamada de *working copy* (OTTE, 2008). Esses conceitos são apresentados na Figura 2.1.

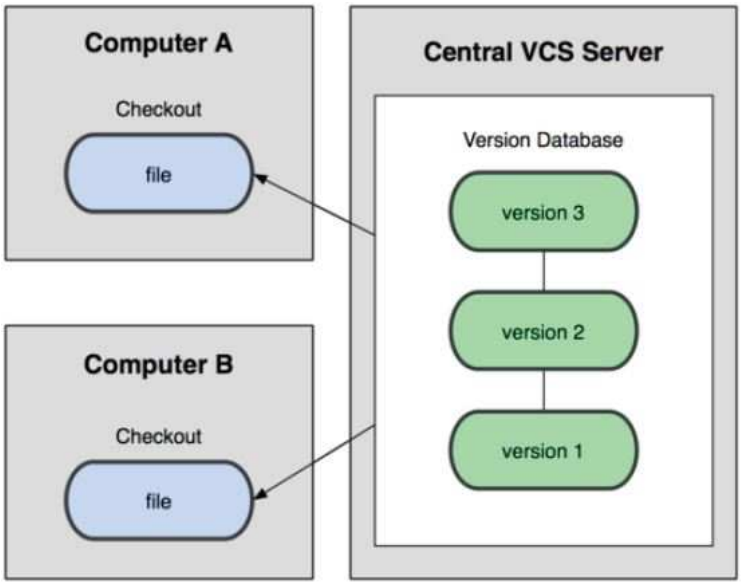

Figura 2.1: Sistema de controle de versão centralizado (CHACON, 2009).

Após as modificações, os arquivos podem ser enviados ao repositório através da operação de *commit*, que inclui uma mensagem com comentários sobre a nova versão. Para atualizar a cópia local, existe a operação de *update*. Quando o servidor possui uma versão mais atual do projeto, o usuário pode juntar com a sua cópia local através de uma

operação de *merge*. Essa operação pode gerar conflitos gerados por diferenças que o VCS não conseguiu juntar que devem ser resolvidos manualmente (OTTE, 2008).

Também é possível a criação de ramos, denominados *branches*, para linhas independentes de desenvolvimento. O ramo principal é chamado de *trunk*. Por fim, o usuário por criar rótulos, denominados *tags*, utilizados para marcar arquivos por relevância ou por alguma característica importante. Cada categoria, *branches*, *tags* e *trunk* é armazenada em uma pasta (OTTE, 2008).

Por muitos anos, o modelo centralizado foi o padrão adotado. A principal desvantagem desse modelo é a dependência do servidor. Se o servidor estiver off-line, não há colaboração entre os desenvolvedores nem atualizações no repositório. Uma vantagem é que o administrador tem um maior controle sobre todas as partes do projeto (CHACON, 2009).

Diferente dos centralizados, os sistemas de controle de versão distribuídos não possuem servidor central, já que todo o repositório, incluindo o histórico, está na máquina do usuário. Uma vantagem é que, apesar de exigir mais espaço em disco, essa estratégia é mais rápida, visto que operações locais são mais eficientes que as remotas. A cópia local é chamada de *local repository*. Repositórios remotos são utilizados apenas para compartilhar o projeto com outro usuário, através das operações de *fetch*, para atualizar sua cópia local, e *push*, para enviar a cópia para o servidor (OTTE, 2008). A Figura 2.2 resume esses conceitos.

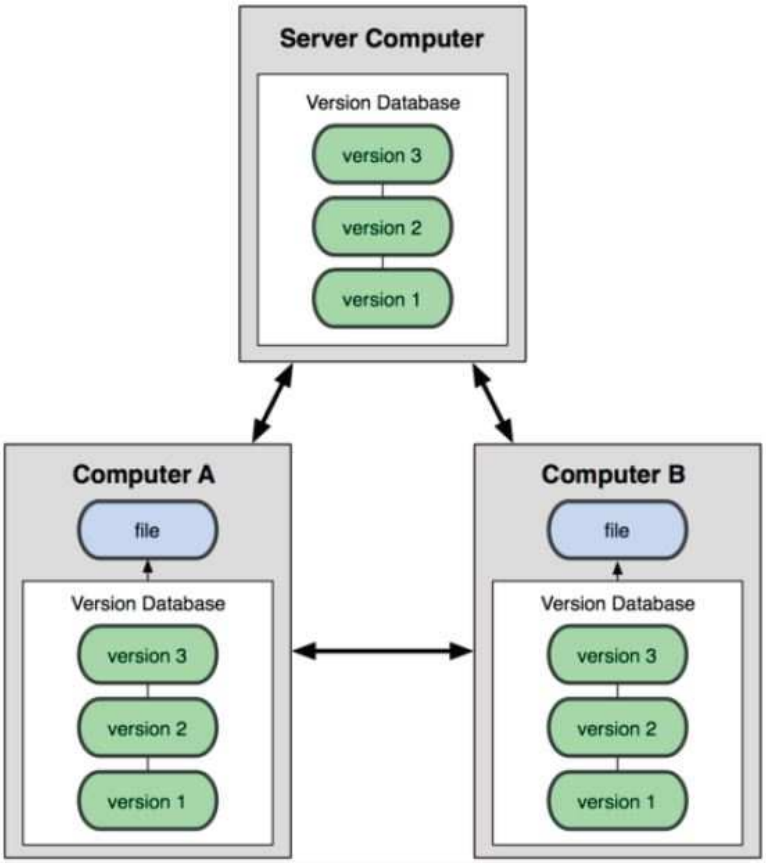

Figura 2.2: Sistema de controle de versão distribuído (CHACON, 2009).

2.2 Subversion

Em 2000, a CollabNet desenvolveu um sistema colaborativo de software chamado CollabNet Enterprise Edition, que incluía um sistema de controle de versão, inicialmente o CVS. Porém, devido a suas limitações, a empresa decidiu desenvolver um novo sistema. O objetivo não era reinventar os conceitos de controle de versão, mas desenvolver um sistema seguindo os conceitos básicos e retirando falhas e limitações do CVS. O Subversion (SVN) foi desenvolvido por Karl Fogel e Ben Collins-Sussman, entre outros. Em 2001, depois de 14 meses, o código-fonte do SVN passou a ser gerenciado pelo próprio SVN, ao invés do CVS (COLLINS-SUSSMAN et al, 2011).

O SVN é um sistema centralizado, portanto segue os conceitos apresentados na Seção 2.1. Cada versão é denominada *revision*, que é o estado dos arquivos a cada *commit* do usuário, gerada com um número único, maior que o da versão anterior. A primeira versão é a de número zero (COLLINS-SUSSMAN et al, 2011). A Figura 2.3 ilustra um repositório, onde cada versão possui uma árvore que representa o estado do repositório depois do *commit*.

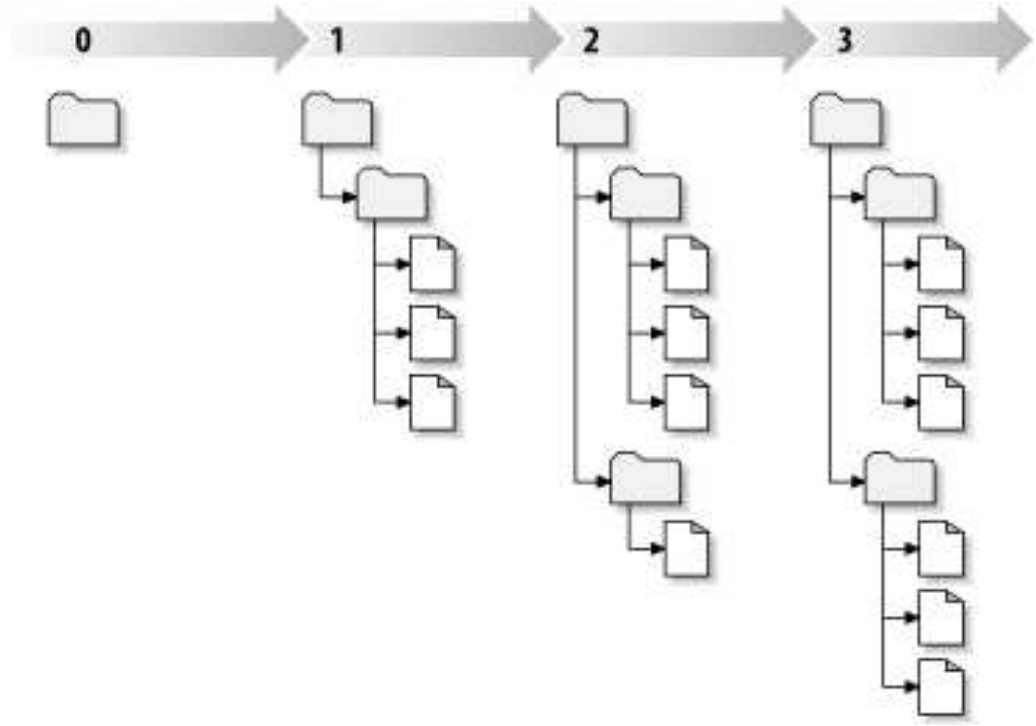

Figura 2.3: Repositório ao longo do tempo (COLLINS-SUSSMAN et al, 2011).

2.3 Git

Em 2002, foi desenvolvido o BitKeeper, um sistema de controle de versão distribuído para o projeto Linux Kernel. Em 2005, a comunidade do projeto rompeu as relações com a empresa que desenvolvia o BitKeeper e a ferramenta deixou de ser gratuita. A comunidade então decidiu criar o seu próprio sistema distribuído, com objetivos como velocidade, interface simples, sólido suporte a múltiplos ramos e capaz de gerenciar grandes projetos como o Linux Kernel (CHACON, 2009). Esse sistema foi chamado de Git.

No Git, cada versão do repositório é denominada *version*. Diferente dos demais sistemas, o Git não armazena o estado do repositório a cada *commit*. O mecanismo do Git salva *snapshots* do estado do repositório e armazena uma referência a esses snapshots, além de não copiar os arquivos não modificados, apenas apontar para a *snapshot* anterior (CHACON, 2009). A Figura 2.4 ilustra esse conceito.

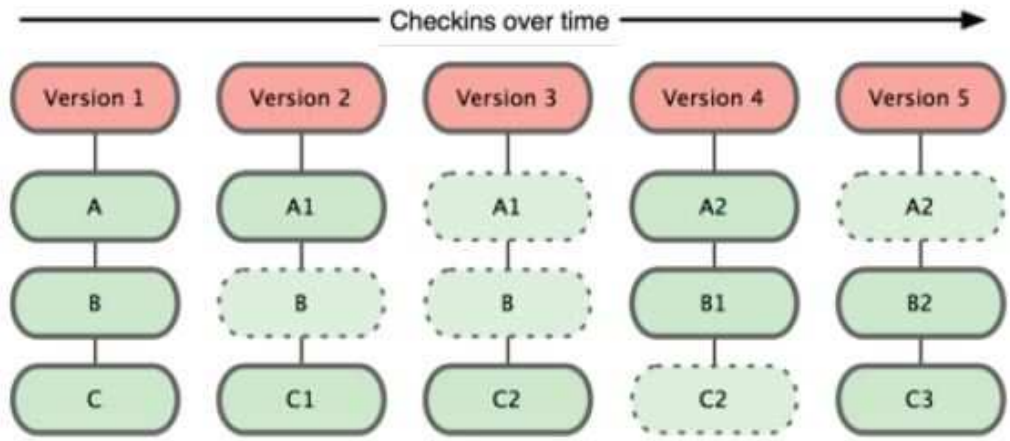

Figura 2.4: Snapshots do repositório ao longo do tempo (CHACON, 2009).

3 PATHFINDER: PRINCIPAIS CARACTERÍSTICAS

Nesse capítulo, será apresentada a abstração proposta para leitura de dados de um sistema de controle de versão genérico. Para isso, primeiro, será identificado o problema que a abstração visa resolver. Em seguida, será apresentada a solução proposta, através de diagrama de classes e a explicação das suas funcionalidades.

3.1 Identificação do problema

Todos os VCS estudados seguem os mesmos princípios de controle de versão, mas eles têm terminologias e mecanismos diferentes, especialmente quando comparado um sistema centralizado a um distribuído, o que dificulta a leitura dos dados dos projetos nos diversos sistemas. Além de ser preciso utilizar programas clientes diferentes, o usuário precisa também se acostumar com os termos distintos e suas finalidades.

3.2 A solução

PathFinder, a solução proposta por este trabalho, é uma API para acesso aos dados de projetos que utilizem algum dos sistemas suportados, sem a necessidade de terminologia específica para cada operação em cada VCS. Para atingir esse objetivo, as funcionalidades dos VCS foram analisadas e divididas nas classes e funcionalidades apresentadas a seguir. A Figura 3.1 apresenta a modelagem completa da API, que será detalhada nas próximas seções. Em todos os diagramas de classes, as operações de *get* e *se*t de atributos foram omitidas.

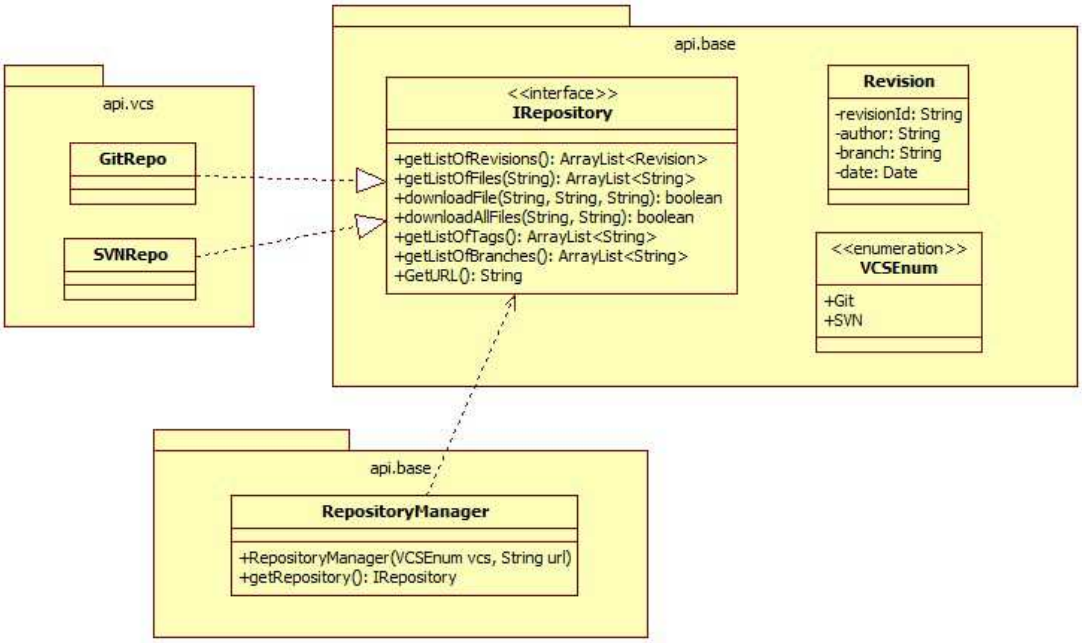

Figura 3.1: Modelagem da API PathFinder

3.2.1 Base

Nesta seção, será apresentado o pacote *api.base*, onde encontra-se a base para a API, ou seja, as principais classes e conceitos para representar um VCS genérico.

3.2.1.1 Revision

O conceito *Revision* foi criado para representar uma revisão. Uma revisão é composta de uma identificação, autor, data e ramo a qual pertence, conforme ilustrado no diagrama de classes na Figura 3.2.

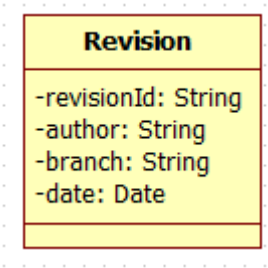

Figura 3.2: Conceito de revisão

3.2.1.2 VCSEnum

A enumeração *VCSEnum* foi criada para especificar qual o VCS está sendo utilizado, conforme ilustrado na Figura 3.3.

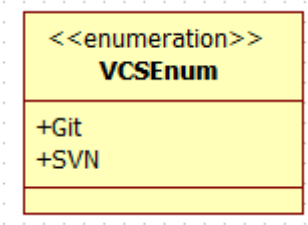

Figura 3.2: Enumeração para especificar o VCS

3.2.1.3 IRepository

A interface *IRepository* foi criada para representar o VCS genérico propriamente dito, de modo a estabelecer um contrato com as classes que implementam cada VCS específico. Nesse contrato, ficam estabelecidas quais as funcionalidades que devem ser oferecidas, seus parâmetros e objetos a serem retornados, conforme ilustra a Figura 3.4.

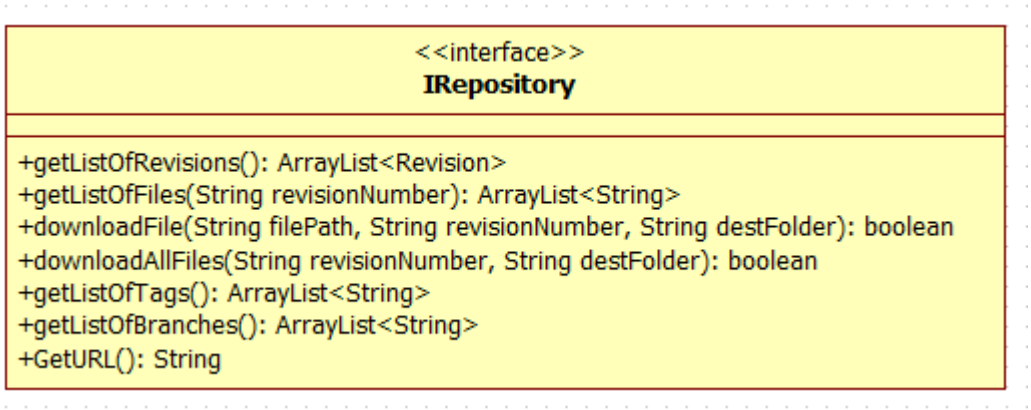

Figura 3.4: Interface para representar o VCS genérico

As funcionalidades são explicadas a seguir:

- GetListOfRevisions: retorna a lista de revisões do repositório;
- GetListOfFiles(revisionNumber): retorna a lista de arquivos do repositório na revisão *revisionNumber*;
- DownloadFile(filePath, revisionNumber, destFolder): salva o arquivo *filePath* da revisão *revisionNumber* na pasta *destFolder*, retornando *true* em caso de sucesso, ou *false* em caso de falha;
- DownloadAllFiles(revisionNumber, destFolder): salva todos os arquivos da revisão *revisionNumber* na pasta *destFolder*, retornando *true* em caso de sucesso, ou *false* em caso de falha;
- GetListOfTags: retorna a lista de tags do repositório;
- GetListOfBranches: retorna a lista de ramos do repositório;
- GetURL: retorna a URL do repositório.

3.2.2 Utils

Nesta seção, será apresentado o pacote *api.utils*, onde encontra-se a parte da API oferecida ao usuário.

3.2.2.1 RepositoryManager

O conceito de RepositoryManager foi criado para oferecer ao usuário uma maneira de acessar os dados do projeto desejado com as operações genéricas, conforme ilustrado na Figura 3.5. O usuário especifica o VCS a ser utilizado apenas na criação do objeto, para que o gerenciador possa configurar o repositório de maneira apropriada.

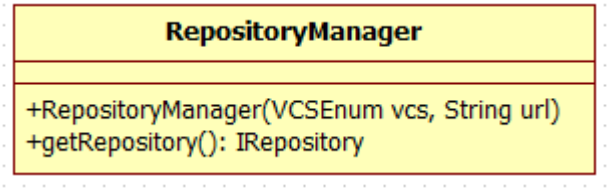

Figura 3.5: Gerenciador de repositório

Para criação do objeto que representa o repositório, o usuário da API inicializa o gerenciador, especificando o VCS, pelo parâmetro do tipo enumeração *vcs*, e a URL, pelo parâmetro do tipo string *url*. Para utilizar as funcionalidades da API, o usuário acessa o repositório pelo método *getRepository*, podendo chamar qualquer um dos métodos estabelecidos pela interface *IRepository*.

3.2.3 VCS

O pacote *api.vcs* é a parte da API que implementa as funções para cada VCS, respeitando o contrato estabelecido através da interface *IRepository*. O pacote é composto pelas classes GitRepo e SVNRepo, que oferecem as implementações para o Git e o SVN, respectivamente.

4 A IMPLEMENTAÇÃO

Neste capítulo, serão apresentados os principais detalhes da implementação da API PathFinder, bem como um exemplo de uso, criado com uma simples interface gráfica. Serão apresentadas as bibliotecas utilizadas para implementar as funcionalidades para os VCS específicos. A API foi implementada em Java, utilizando o ambiente do Eclipse, versão Indigo Service Release 2.

A implementação da classe *Revision* será omitida porque é composta apenas dos atributos e dos métodos *get* e *set* para cada um deles. Também será omitida a implementação do gerenciador, visto que o código é composto apenas da inicialização do repositório e do acesso ao mesmo. A enumeração *VCSEnum* não possui implementação, apenas a declaração.

4.1 As bibliotecas

A biblioteca utilizada para implementar as funcionalidades para o SVN foi a SVNKit, versão 1.3.8. A biblioteca SVNKit, independente de sistema operacional, Open Source e implementada puramente em Java, oferece funcionalidades para acessar e manipular repositórios no SVN (SVNKit 2013).

A biblioteca utilizada para implementar as funcionalidades para o Git foi a JGit, versão 3.0.0. A biblioteca JGit, Open Source e utilizada em produtos como Eclipse e NetBeans, oferece funcionalidades para acesso a repositórios no Git, bem como interfaces para executar os principais comandos (JGit 2013).

4.2 Funcionalidades para o SVN

As funcionalidades para o SVN foram implementadas na classe SVNRepo, no pacote *api.vcs*, utilizando a biblioteca SVNKit. Como o SVN é um VCS centralizado, todas as operações são feitas através de acesso remoto, sem a necessidade de salvar uma cópia do repositório na máquina local.

No SVN não existem ramos e tags propriamente ditos. O esquema do SVN consiste em três pastas para armazenar esses conceitos: *trunk*, *branches* e *tags*. Ao criar um ramo, o repositório é copiado para uma pasta com o nome do ramo dentro da pasta branches. O mesmo acontece com as *tags*, na respectiva pasta. Por isso, a classe SVNRepo contém constantes para armazenar o nome dessas pastas.

4.2.1 Inicialização

A inicialização do repositório no SVN trata-se basicamente da inicialização dos atributos necessários para a manipulação do mesmo. A classe possui os seguintes atributos: uma string que armazena a URL, o gerenciador do SVNKit e um lista de objetos do tipo *SVNInfo*, onde cada objeto representa um arquivo ou pasta do repositório.

4.2.2 Lista de revisões

Para buscar a lista de revisões, dada a URL do projeto, foi utilizada a operação de *log* do SVNKit, que busca todas as operações de *commit* e seus atributos, como autor, data e mensagem.

4.2.3 Lista de arquivos

Para buscar a lista de arquivos, foi utilizada a operação de *doInfo* do SVNKit, que busca informações úteis sobre os itens do repositório em uma determinada revisão. Essa operação recebe um handler do tipo *ISVNInfoHandler* que permite tratar cada item através do método *handleInfo()*. Nesse método, cada item é colocado na lista armazenada na classe SVNRepo. No final, essa lista é percorrida para retornar a lista de URL dos arquivos.

4.2.4 Lista de ramos e tags

Como os conceitos de ramos e tags são representadas por cópias de um determinado estado do repositório nas pastas *branches* e *tags*, respectivamente, para listar ramos e tags é necessário apenas percorrer essas pastas e retornar os nomes das pastas dentro delas. Para isso, foi utilizada a operação de *doList*, que busca informações úteis sobre os diretórios filhos. Essa operação recebe um handler do tipo *ISVNDirEntryHandler* que permite tratar cada item através do método *handleDirEntry*. Nesse método, a URL relativa de cada entrada é armazenada em uma lista a ser retornada como *ramo* ou *tag*.

4.2.5 Download de arquivos

Para o download de um arquivo, a lista armazenada na classe SVNRepo é preenchida com o mesmo algoritmo explicado na seção 4.2.3. Com o objeto *SVNInfo* do arquivo a ser salvo, a operação *doGetFileContents* do SVNKit é utilizada para criar o arquivo na pasta de destino especificada.

Para o download de todos os arquivos, a lista é novamente preenchida com o mesmo algoritmo da seção 4.2.3, para então ser percorrida. Para cada item da lista, o algoritmo verifica o tipo do item. Para arquivos, a função para download de um arquivo é utilizada. Para pastas, é criada uma nova pasta no diretório de destino, com o mesmo nome e na mesma posição hierárquica em relação as demais pastas no repositório.

4.3 Funcionalidades para o Git

As funcionalidades para o Git foram implementadas na classe GitRepo, no pacote *api.vcs*, utilizando a biblioteca JGit. Como o Git é um VCS distribuído e trabalha essencialmente em cópias locais, a biblioteca JGit não oferece funcionalidades remotas, exceto para operações essencialmente remotas, como *fetch* e *push*. Por isso, para acesso as informações do repositório, o mesmo precisa ser copiado para a máquina local.

4.3.1 Inicialização

A inicialização do repositório do Git trata-se da inicialização da variável do tipo string que armazena a URL e da cópia do repositório para um diretório temporário, através da operação de *clone* da JGit. O diretório temporário é deletado caso exista, para então ser clonado.

4.3.2 Lista de revisões

Para listar as revisões do repositório, a operação de log da JGit é utilizada. Ela retorna uma lista de objetos do tipo *RevCommit*, que é percorrida para gerar a lista de revisões a ser retornada.

4.3.3 Lista de arquivos

Para listar os arquivos de uma determinada revisão, é criado um objeto do tipo *TreeWalk*, que representa o repositório na biblioteca JGit. A árvore é percorrida para preencher a lista de URL dos arquivos a ser retornada.

4.3.4 Lista de ramos e tags

Para listar os ramos e tags do repositório, são utilizadas as operações de *branchList* e *tagList*, respectivamente. Essa operações retornam uma lista de objetos *Ref*, que contém os nomes dos ramos ou tags a serem retornados.

4.3.5 Download de arquivos

Para o download de um arquivo, o algoritmo cria a árvore conforme explicado na seção 4.3.3 e faz uma busca para encontrar o objeto que representa o arquivo. Encontrado o objeto, o arquivo é salvo na pasta de destino. Para o download de todos os arquivos, a árvore é criada e percorrida por completo para salvar cada arquivo na pasta de destino correspondente, respeitando a hierarquia de pastas no repositório.

4.4 Extensão da API

Para extender as funcionalidades da API para um determinado sistema de controle de versão, sugere-se que o desenvolvedor inclua uma classe no pacote *api.vcs* para adicionar a implementação específica para o sistema, além de incluir as bibliotecas necessárias no projeto. Também é preciso incluir um campo na enumeração *VCSEnum*, do pacote *api.base*, com o nome do sistema, e garantir que o gerenciador *RepositoryManager*, do pacote *api.utils*, inclua o tratamento para esse sistema.

4.5 Exemplo de uso

Para demonstrar o uso da API, foi implementada uma interface gráfica simples, apenas para executar os comandos e exibir os resultados na tela, conforme Figura 4.1. A URL do projeto e o VCS utilizado devem ser especificados para então o repositório ser configurado ao clicar no botão “Set Repository”. Em seguida, é possível escolher a operação a ser executada, preencher os parâmetros necessários e executá-la ao clicar no botão “GO!”. O resultado é exibido na área de texto. Exemplos das operações e seus resultados são exibidos na Figura 4.2 e Figura 4.3. Para testar a API, projetos de teste foram criados para o SVN e Git, conforme Apêndices A e B, respectivamente.

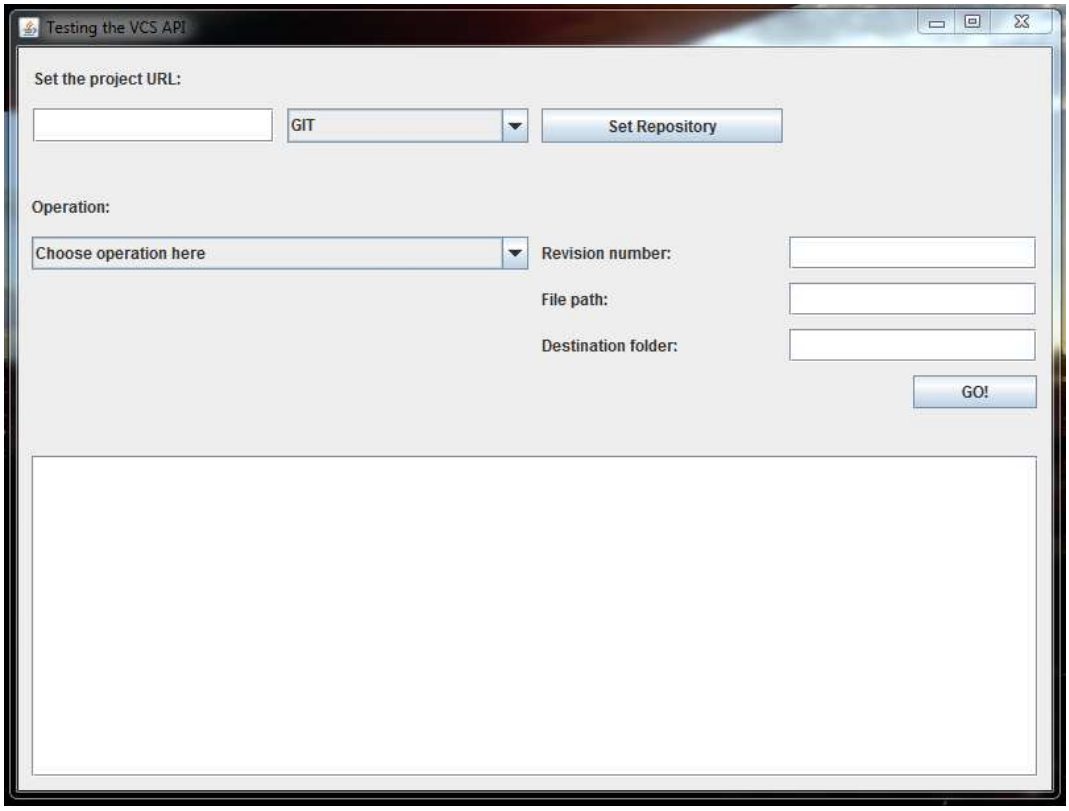

Figura 4.1: Interface gráfica

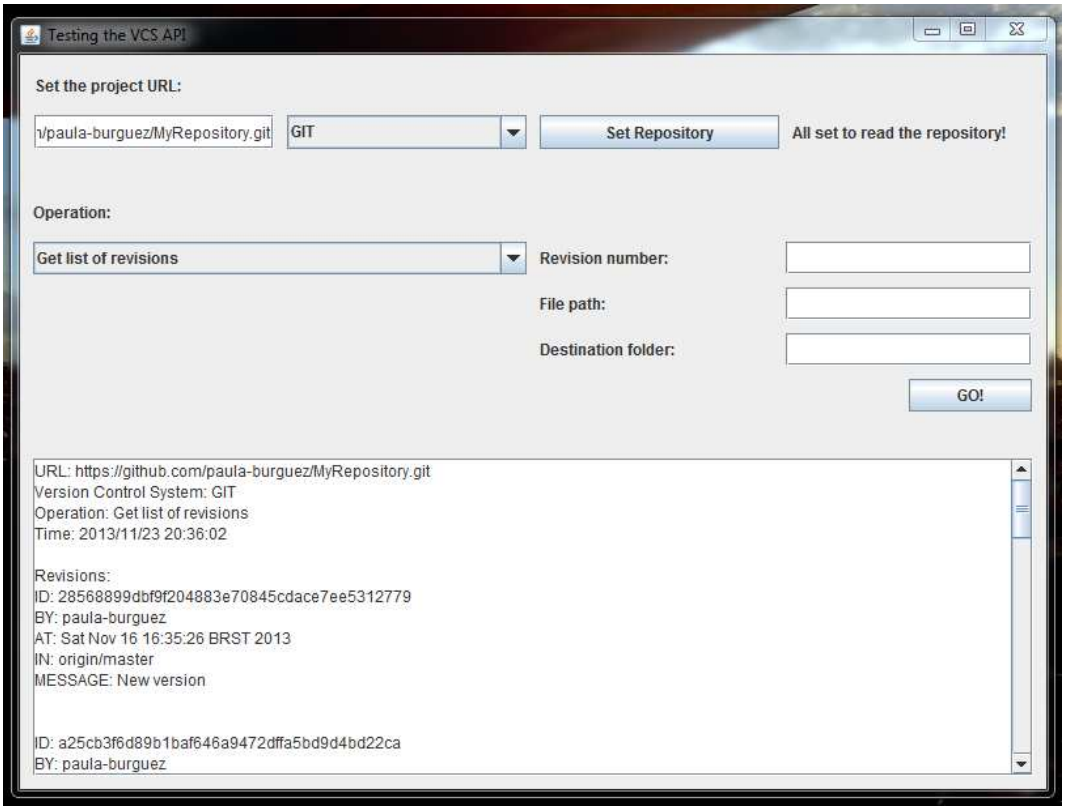

Figura 4.2: Resultado da operação de GetListOfRevisions

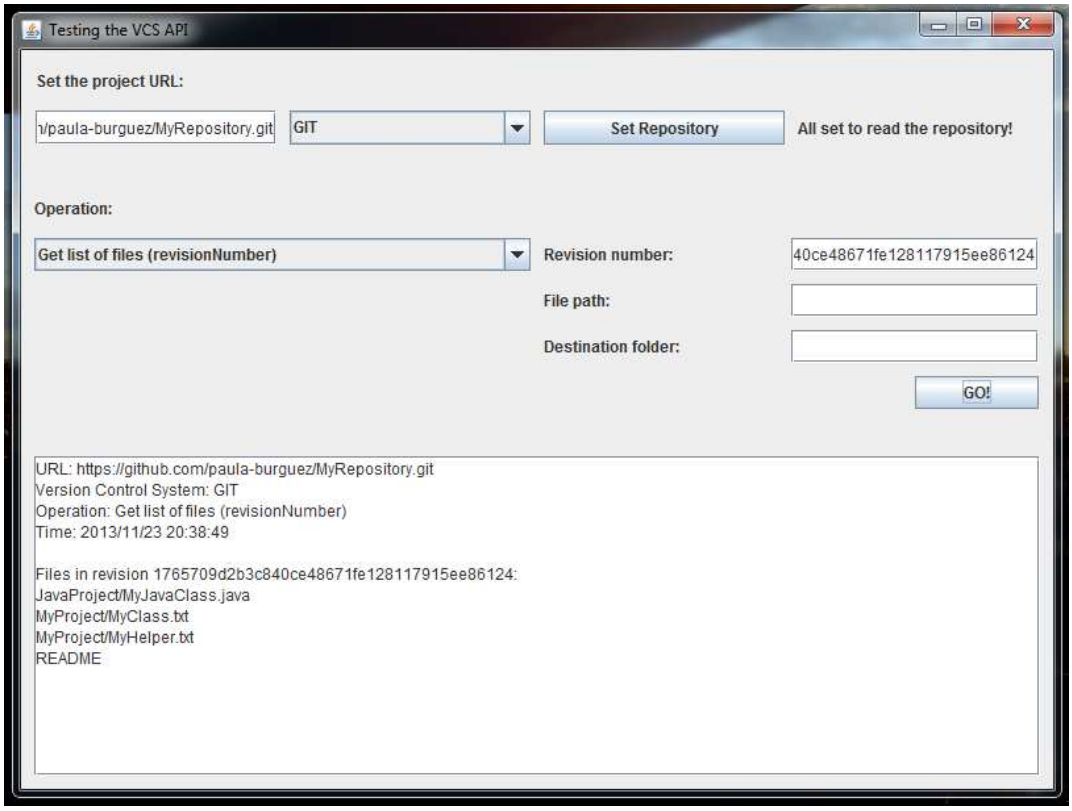

Figura 4.3: Resultado da operação de GetListOfFiles, com uma revisão específica

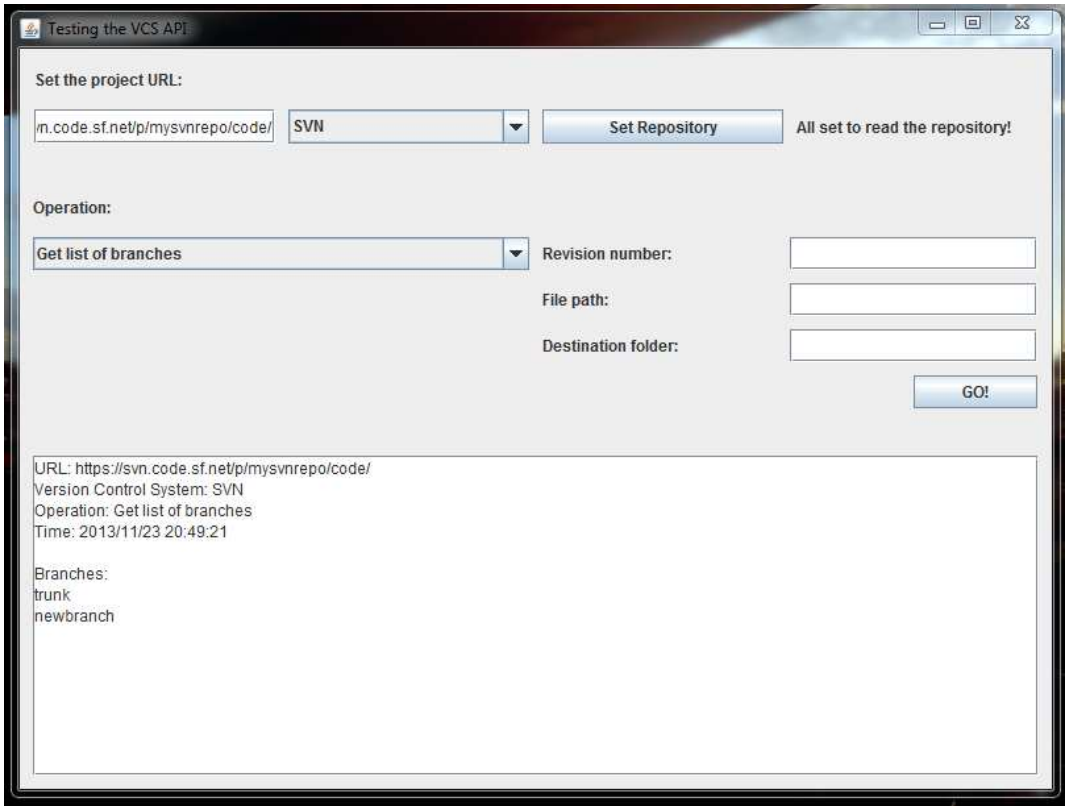

Figura 4.4: Resultado da operação de GetListOfBranches

5 ESTUDO DE CASO

Nesse capítulo, será apresentado o estudo de caso para demonstrar o funcionamento da API com projetos de pequeno, médio e grande porte. Serão apresentados os projetos escolhidos, para então coletar as informações de versionamento pelas ferramentas de linha de comando originais do SVN e Git, e compará-las com as informações coletadas pela API desenvolvida nesse trabalho.

5.1 Projetos escolhidos

Os projetos escolhidos foram divididos em três categorias: projetos de pequeno porte, com menos de mil arquivos com código, projetos de médio porte, com até 3 mil arquivos, e projetos de grande porte, com mais de 3 mil arquivos. Essas categorias foram definidas pela observação do tamanho médio dos projetos encontrados, sem critérios específicos. A amostra utilizada nessa etapa do trabalho foi de 3 projetos pequenos, 3 médios e 3 grandes, escolhidos por uma busca na internet.

Os projetos de pequeno porte escolhidos para o SVN foram: HtmlCleaner, WYSIWYG Calculator e Java OCR. Como projetos de médio porte, foram selecionados: Notepad++, PHP QR Code e Naanal PHP Framework. Por fim, os projetos de grande porte utilizados foram: PDFCreator, jEdit e OpenOffice. Para cada projeto, a URL utilizada para acessá-lo é apresentada na Tabela 5.1.

Tabela 5.1: URL dos projetos para o SVN

HtmlCleaner	http://svn.code.sf.net/p/htmlcleaner/code
WYSIWYG Calculator	http://svn.code.sf.net/p/wysiwygcalc/code
Java OCR	http://svn.code.sf.net/p/javaocr/code
Notepad++	http://svn.code.sf.net/p/notepad-plus/code/
PHP QR Code	http://svn.code.sf.net/p/phpqrcode/code/
Naanal PHP Framework	http://svn.code.sf.net/p/naanalframework/code
PDFCreator	http://svn.code.sf.net/p/pdfcreator/code/
jEdit	https://jedit.svn.sourceforge.net/svnroot/jedit/jEdit
OpenOffice	https://svn.apache.org/repos/asf/openoffice/

Já para o Git, os projetos de pequeno porte escolhidos foram: GitHub Android App, PHPExcel e Facebook SDK for Android. Como projetos de médio porte, foram utilizados os seguintes: Homebrew, Three.js e Symfony PHP Framework. Por fim, os projetos de grande porte escolhidos para o Git foram: Spring Framework, Hibernate's Core Object/Relational Mapping Functionality e Node.js. Para cada projeto, a URL utilizada para acessá-lo é apresentada na Tabela 5.2.

Tabela 5.2: URL dos projetos para o Git

GitHub Android App	https://github.com/github/android.git
PHPExcel	https://github.com/PHPOffice/PHPExcel.git
Facebook SDK for Android	https://github.com/facebook/facebook-android-sdk.git
Three.js	https://github.com/mrdoob/three.js.git
Homebrew	https://github.com/mxcl/homebrew.git
Symfony PHP Framework	https://github.com/symfony/symfony.git
Spring Framework	https://github.com/spring-projects/spring-framework.git
Hibernate ORM	https://github.com/hibernate/hibernate-orm.git
Node.js	https://github.com/joyent/node.git

5.2 Coleta de informações sobre versionamento

Nessa etapa, para demonstrar que as informações foram coletadas corretamente, foram usados os comandos originais do SVN e Git para obter essas informações e compará-las com as coletadas pela API. Os testes para demonstrar o correto funcionamento da API foram feitos em todos os projetos escolhidos, porém nem todos os resultados serão exibidos devido a grande quantidade de dados.

Os comandos originais para o SVN e Git que foram executados são exibidos na Tabela 5.3, onde *url* refere-se a URL do projeto conforme explicado na seção 5.1, *arquivo* refere-se ao arquivo de saída onde serão salvas as informações, *destino* refere-se a pasta de destino, e *rev* a revisão. No caso do Git, o projeto deve ser clonado, utilizando o comando *git clone*, e as demais operações devem ser feitas dentro da pasta local do projeto. Alguns resultados são apresentados nas Figura 5.1, Figura 5.2, Figura 5.3 e Figura 5.4, enquanto outros podem ser encontrados no Apêndice D. Todos os resultados demonstram o correto funcionamento da API para as funcionalidades de coleta de informações sobre versionamento.

Tabela 5.3: Comandos originais executados para listas

	SVN	*Git*
Lista de revisões	svn log url > arquivo	git log > arquivo
Lista de ramos	svn ls url/branches –v > arquivo	git branch –r > arquivo
Lista de tags	svn ls url/tags –v > arquivo	git tag > arquivo
Lista de arquivos	svn co –r rev url destino cd destino; ls –R > arquivo	git checkout rev ls –R > arquivo

```
1542315 hdu                  Nov 15 14:37 ./
1332937 jsc                  May 02  2012 AOO340/
1376039 jsc                  Aug 22  2012 AOO341/
1513392 jsc                  Aug 13 06:09 AOO400/
1526173 hdu                  Sep 25 10:35 AOO401/
1542315 hdu                  Nov 15 14:37 SNAPSHOT/
```

Figura 5.1: Lista de tags com comando do SVN para OpenOffice

```
Tags:
AOO340
AOO341
AOO400
AOO401
SNAPSHOT
```

Figura 5.2: Lista de tags com a API para OpenOffice

```
origin/HTMLReader
origin/OOCalcReader
origin/autoShapes
origin/autofilter
origin/calcEngine
origin/cellRestructure
origin/charts
origin/develop
origin/develop_1.7.9
origin/develop_2.0.0
origin/experimental
origin/master
origin/style
```

Figura 5.3: Lista de ramos com comando do Git para PHPExcel

```
Branches:
origin/HTMLReader
origin/OOCalcReader
origin/autoShapes
origin/autofilter
origin/calcEngine
origin/cellRestructure
origin/charts
origin/develop
origin/develop_1.7.9
origin/develop_2.0.0
origin/experimental
origin/master
origin/style
```

Figura 5.4: Lista de ramos com a API para PHPExcel

5.3 Download de arquivos

Assim como na etapa anterior, foram usados os comandos originais do SVN e Git para obter os arquivos e compará-los com os coletados pela API. Os testes para demonstrar o correto funcionamento da API foram feitos em todos os projetos escolhidos, porém nem todos os resultados serão exibidos devido a grande quantidade de dados. Os comandos originais para o SVN e Git que foram executados são exibidos na Tabela 5.4, com as mesmas variáveis que a Tabela 5.3. O download de um arquivo ou de todos foi feito da mesma maneira, visto que basta modificar a *url*.

Em relação ao SVN, os resultados são apresentados na Tabela 5.5, enquanto a Tabela 5.6 apresenta os resultados para o Git. Para o SVN, o tempo da linha de comando, *Tempo LC* na tabela, inclui o download da pasta de configurações, a *.svn*, cujo tamanho não está incluído no campo *tamanho* da tabela. As pastas de ambas as fontes foram comparadas para verificar que possuem os mesmos arquivos e subpastas. Todos os testes demonstram o correto funcionamento da API para as funcionalidades de download de arquivos.

Tabela 5.4: Comandos originais executados para download

	SVN	*Git*
Download	svn co –r rev URL destino	git checkout rev

Tabela 5.5: Resultados dos downloads para o SVN

	Revisão	*Tamanho*	*Tempo LC*	*Tempo API*
WYSIWYG Calculator	10	157 KB	1s	1min50s
Java OCR	34	2.05MB	5s	3min32s
HtmlCleaner	99	3.56MB	7s	5min24s

Tabela 5.6: Resultados dos downloads para o Git

	Revisão	*Tamanho*	*Tempo LC*	*Tempo API*
Node.js	887f05692353d01ec97e1e99ff34479a433ad985	12.7MB	8s	2s
Spring Framework	823dbdf232dbd5d88391f88c6e390ce51a3852aa	37.4MB	4s	16s
Hibernate	355e81a7f11cfad11f46408a481c7ed090174080	76MB	36s	18s

Um ponto importante observado com os testes é que, apesar de ser a operação mais próxima de um download por linha de comando, a operação de *checkout* no Git não é ideal para uma comparação. Isso ocorre porque API faz uma cópia de todos os arquivos para a pasta destino, enquanto o Git apenas atualiza os arquivos que forem necessários

de acordo com a versão desejada. Esse fator foi observado no projeto Spring Framework, visto que a revisão era bem próxima da atual, a *head*, reduzindo o tempo já que poucos arquivos precisaram ser atualizados.

6 CONCLUSÃO

Neste trabalho foi criada uma abstração para sistemas de controle de versão, com o objetivo de oferecer acesso aos arquivos e as informações de versionamento de um projeto independente do sistema. Para isso, as principais funcionalidades dos sistemas, tanto distribuídos quanto centralizados, foram analisadas para definir os dados mais importantes a serem extraídos.

Então, uma interface foi criada para representar um sistema genérico. Foram utilizadas bibliotecas dos sistemas específicos em Java para implementar o acesso aos dados do projeto do SVN e Git. Uma interface gráfica simples foi desenvolvida para realizar os primeiros testes com projetos pequenos, criados especialmente para este trabalho. Em seguida, projetos de código aberto, de portes pequeno, médio e grande, foram escolhidos para a realização de experimentos que demonstraram o correto funcionamento da API.

Apesar da pouca documentação das bibliotecas para sistemas específicos, como a SVNKit e a JGit utilizadas no trabalho, foi observado que é possível implementar funcionalidades que as ferramentas de linha de comando também oferecem. Porém, a API PathFinder possui algumas limitações, como o desempenho em relação ao tempo de processamento, e a necessidade de recompilação do projeto se for adicionado o suporte para outros sistemas.

Em relação a trabalhos futuros, seria interessante expandir essa funcionalidades. Além disso, poderiam ser implementadas as mesmas funcionalidades para outros sistemas de controle de versão, como o CVS e Mercurial. Outro aspecto que poderia ser explorado é a utilização de padrões de projeto, especialmente em favor do reuso da API para, por exemplo, oferecer suporte a outros sistemas de controle de versão.

REFERÊNCIAS

BALL, T; PORTER, J. K. A.; SIY, H. P. If Your Version Control System Could Talk. **ICSE ’97 Workshop on Process Modeling and Empirical Studies of Software Engineering**, maio 1997.

CHACON, S. Pro Git. Apress, Jun 2009.

CHERAIT, H; BOUNOUR, N. Modeling Software Evolution through Version Control System. **Department of Computer Science Badji Mokhtar, Annaba University**, 2012.

COLLINS-SUSSMAN, B.; FITZPATRICK, B. W.; PILATO, C. M. Version Control with Subversion, for Subversion 1.7, 2011.

Facebook SDK for Android Project. Disponível em <https://github.com/facebook/facebook-android-sdk>. Acessado em 02/12/2013.

GitHub Android App Project. Disponível em <https://github.com/github/android>. Acessado em 02/12/2013.

GitHub Help. Disponível em <https://help.github.com/articles/>. Acessado em 03/11/2013.

Hibernate's Core Object/Relational Mapping Functionality Project. Disponível em <https://github.com/hibernate/hibernate-orm>. Acessado em 02/12/2013.

Homebrew Project. Disponível em <https://github.com/mxcl/homebrew>. Acessado em 02/12/2013.

HtmlCleaner Project. Disponível em <http://sourceforge.net/projects/htmlcleaner/>. Acessado em 02/12/2013.

Java OCR Project. Disponível em <http://sourceforge.net/projects/javaocr/>. Acessado em 02/12/2013.

jEdit Project. Disponível em <http://sourceforge.net/projects/jedit/>. Acessado em 02/12/2013.

JGit Website. Disponível em <http://www.eclipse.org/jgit/>. Acessado em 19/08/2013.

Kleine, M; Hirschfeld, R; Bracha, G. An Abstraction for Version Control Systems. **IT Systems Engineering, Universität Potsdam**, Alemanha, 2012.

Node.js Project. Disponível em <https://github.com/joyent/node>. Acessado em 02/12/2013.

Notepad++ Project. Disponível em <http://sourceforge.net/p/notepad-plus/>. Acessado em 02/12/2013.

OpenOffice Project. Disponível em <http://openoffice.apache.org/source.html>. Acessado em 02/12/2013.

OTTE, S. Version Control Systems. **Institute of Computer Science, Freie Universität Berlin**, Alemanha, set. 2008.

PDFCreator Project. Disponível em <http://sourceforge.net/projects/pdfcreator/>. Acessado em 02/12/2013.

PHPExcel Project. Disponível em <https://github.com/PHPOffice/PHPExcel>. Acessado em 02/12/2013.

PHP QR Coce Project. Disponível em <http://sourceforge.net/projects/phpqrcode/>. Acessado em 02/12/2013.

Spring Framework Project. Disponível em <https://github.com/spring-projects/spring-framework>. Acessado em 02/12/2013.

SVNKit Website. Disponível em <http://svnkit.com/>. Acessado em 19/08/2013.

Symfony PHP Framework Project. Disponível em <https://github.com/symfony/symfony>. Acessado em 02/12/2013.

Three.js Project. Disponível em < https://github.com/mrdoob/three.js/>. Acessado em 02/12/2013.

WYSIWYG Calculator Project. Disponível em <http://sourceforge.net/projects/wysiwygcalc/>. Acessado em 02/12/2013.

APÊNDICE A - Instalação do SVN e criação de exemplo

Esse apêndice destina-se a explicar os passos para instalação do SVN e sua aplicação em um exemplo. O projeto de exemplo criado serviu para testes da API desenvolvida nesse trabalho.

A.1 Instalação

A seguir são explicados os passos para o uso de um ambiente do SVN no Windows 7, bem como a utilização da plataforma SourceForge para compartilhamento de projetos.

1. Instalar Subversion para Windows, versão 1.8.4;
2. Criar uma conta no SourceForge;
3. Instalar PuTTYGen e Plink para geração da chave SSH.

A.2 Criação do exemplo

A seguir são explicados alguns passos para a criação de um projeto simples de teste, considerando o ambiente do SVN no Windows foi instalado conforme seção anterior.

1. Criar projeto no SourceForge;
2. Gerar chave SSH com o PuTTY e adicionar ao projeto no SourceForge. Utilizar username@svn.code.sf.net como *key comment*;
3. No prompt de comando do SVN, geralmente encontrado na pasta C:\Arquivo de Programas\Subversion\bin:
    i. Checkout no projeto: svn checkout url dir;
    ii. Para adicionar arquivos ou pastas, usar svn add;
    iii. Para enviar as alterações, utilizar svn ci –m “mensagem”.

APÊNDICE B - Instalação do Git e criação de exemplo

Esse apêndice destina-se a explicar os passos para instalação do Git e sua aplicação em um exemplo. O projeto de exemplo criado serviu para testes da API desenvolvida nesse trabalho.

B.1 Instalação

A seguir são explicados os passos para a instalação do ambiente do Git no Windows 7, bem como a utilização da plataforma GitHub para compartilhamento de projetos.

1. Criar de uma conta no http://github.com;
2. Seguir instruções em http://help.github.com/set-up-git-redirect. Para o trabalho, foi instalado o GitHub para Windows.

B.2 Criação do exemplo

A seguir são explicados alguns passos para a criação de um projeto simples de teste, considerando que uma conta já foi criada no GitHub e que ambiente do Git no Windows foi instalado conforme seção anterior. O exemplo foi criado seguindo instruções nos artigos encontrados na seção do GitHub Help.

1. No Git Shell, configurar nome e email para serem utilizados nos commits:
    i. Nome: git config --global user.name "nome"
    ii. Email: git config --global user.email "email"
2. No Git Shell, criar chave SSH:
    i. Criar chave: ssh-keygen -t rsa -C "email";
    ii. Abrir o arquivo id_rsa.pub e copiar o conteúdo;
    iii. Adicionar o conteúdo nas *SSH Keys* em *Accounts Settings* no GitHub.
3. Na interface do Github, criar um novo repositório;
4. No Git Shell, criar projeto inicial:
    i. Criar o diretório com o mesmo nome: mkdir MyRepository;
    ii. Entrar no diretório: cd MyRepository;
    iii. Inicializar diretório: git init;

iv. Criar o arquivo README: touch README.

5. No Git Shell, realizar operação de commit no estado atual do repositório:

i. Adicionar arquivo README no projeto: git add README;

ii. Commit no arquivo: git commit –m “adding README file”.

6. No Git Shell, conectar repositório local a conta no GitHub, onde URL é o caminho no formato https://github.com/username/MyRepository.git:

i. Configurar repositório remoto: git remote set-url origin url;

ii. Especifica o ramo: git push origin master.

7. Para novos arquivos ou modificações em arquivos já existentes, seguir os passos no item 5. Para modificações em arquivos já adicionados, utilizar a instrução 5 (ii) com –a no final. Para enviar as alterações para o servidor, executar git push.

APÊNDICE C - Códigos

Esse apêndice é destinado ao código desenvolvido para o trabalho. Em todas as classes, comentários, linhas do tipo *import* e métodos dos tipos *get* e *set* de parâmetros da classe são omitidos.

Código C.1: VCSEnum

```
package api.base;

public enum VCSEnum {
	SVN, GIT;
}
```

Código C.2: Revision

```
public class Revision {
	private String revisionId;
	private String author;
	private String branch;
	private String message;
	private Date date;
}
```

Código C.3: VCSEnum

```
package api.base;

public interface IRepository {

	public ArrayList<Revision> getListOfRevisions() throws Exception;
	public ArrayList<String> getListOfFiles(String revisionNumber) throws
Exception;
	public boolean downloadFile(String filePath, String revisionNumber,
String destFolder) throws Exception;
	public boolean downloadAllFiles(String revisionNumber, String
destFolder) throws Exception;
	public ArrayList<String> getListOfTags() throws Exception;
	public ArrayList<String> getListOfBranches() throws Exception;
	public String getURL();

}
```

Código C.4: RepositoryManager

```
package api.utils;

public class RepositoryManager {
	private IRepository repo;

	public RepositoryManager(VCSEnum vcs, String url) throws Exception {
		switch(vcs)
		{
			case SVN: repo = new SVNRepo(url); break;
			case GIT: repo = new GitRepo(url); break;
			default: repo = null;
		}
	}
}
```

Código C.5: GitRepo

```
package api.vcs;

public class GitRepo implements IRepository {
	private String path;
	private TreeWalk repoInfo;
	private String tempDir;

	public GitRepo(String path) throws Exception {

		this.path = path;
		this.tempDir = System.getProperty("java.io.tmpdir") + "\\" +
"gittemp\\";

		cloneRepo();
	}

	/* INTERFACE METHODS - BEGIN */

	@Override
	public ArrayList<Revision> getListOfRevisions() throws Exception {
		ArrayList<Revision> list = new ArrayList<>();

		Git git = Git.open(new File(tempDir));

	    Iterable<RevCommit> commits = git.log().all().call();
	    Repository repo = git.getRepository();

	    for (RevCommit commit : commits)
	    {
	       Revision rev = new Revision();
	       rev.setRevisionId(commit.getName());
	       rev.setAuthor(commit.getCommitterIdent().getName());
	       rev.setDate(new Date(commit.getCommitTime() * 1000L));
	       rev.setBranch(getBranchOfCommit(repo, commit));
	       rev.setMessage(commit.getFullMessage());
	       list.add(rev);
	    }

		return list;
```

```
	}

	@Override
	public ArrayList<String> getListOfFiles(String revisionNumber) throws
Exception {
		return getRevision(revisionNumber);
	}

	@Override
	public boolean downloadFile(String filePath, String revisionNumber,
String destFolder) throws Exception {
		ArrayList<String> listOfFiles = getRevision(revisionNumber);
		int index = listOfFiles.indexOf(filePath);

		if (index != -1) {
			Git git = Git.open(new File(tempDir));
		    Repository repo = git.getRepository();

		    repoInfo = getNewWalkTree(revisionNumber, repo);
		    repoInfo.next();

		    int i = 0;
		    while (i != index)
		    {
			repoInfo.next();
			i++;
		    }

		    String filename = filePath.replace(path, "");
			index = filename.lastIndexOf("/");
			filename = index != -1 ? filename.substring(index + 1,
filename.length()) : filename;

			ObjectId objectId = repoInfo.getObjectId(0);
		    ObjectLoader loader = repo.open(objectId);
			OutputStream output = new FileOutputStream(destFolder +
"\\" + filename);
		    loader.copyTo(output);

		    return true;
		}
		else return false;
	}

	@Override
	public boolean downloadAllFiles(String revisionNumber, String
destFolder) throws Exception {
		Git git = Git.open(new File(tempDir));
	    Repository repo = git.getRepository();

	    repoInfo = getNewWalkTree(revisionNumber, repo);

	    while (repoInfo.next())
	    {
		String srcPath = repoInfo.getPathString().replace(path, "");

		int index = srcPath.lastIndexOf("/");
```

```
                    String folders = index != -1 ? folders = ((String)
srcPath.subSequence(0, index)) : "";
                    createDirIfDoesNotExist(destFolder + "\\" + folders +
"\\");

             String filename = repoInfo.getPathString().replace(path, "");
                    index = filename.lastIndexOf("/");
                    filename = index != -1 ? filename.substring(index + 1,
filename.length()) : filename;

                    ObjectId objectId = repoInfo.getObjectId(0);
                ObjectLoader loader = repo.open(objectId);

                    OutputStream output = new FileOutputStream(destFolder +
"\\" + folders + "\\" + filename);

                loader.copyTo(output);

          }

          return true;
      }

      @Override
      public ArrayList<String> getListOfTags() throws Exception {
            ArrayList<String> list = new ArrayList<>();

            Git git = Git.open(new File(tempDir));
          List<Ref> tags = git.tagList().call();

          for (Ref ref : tags)
            list.add(Repository.shortenRefName(ref.getName()));

            return list;
      }

      public ArrayList<String> getListOfBranches() throws Exception {
            ArrayList<String> list = new ArrayList<>();

            Git git = Git.open(new File(tempDir));
            List<Ref> branches =
git.branchList().setListMode(ListMode.REMOTE).call();

          for (Ref branch : branches)
            list.add(Repository.shortenRefName(branch.getName()));

            return list;
      }

      @Override
      public String getURL() {
            return this.path;
      }

      /* INTERFACE METHODS - END */

      /* PRIVATE METHODS - BEGIN */
```

```
	private ArrayList<String> getRevision(String revision) throws
Exception {
		ArrayList<String> list = new ArrayList<>();

		Git git = Git.open(new File(tempDir));
	    Repository repo = git.getRepository();

		repoInfo = getNewWalkTree(revision, repo);

	    while(repoInfo.next())
		list.add(repoInfo.getPathString());

	    return list;
	}

	private TreeWalk getNewWalkTree(String revision, Repository repo)
throws Exception {
		ObjectId lastCommitId = repo.resolve(revision.equals("0") ?
Constants.HEAD : revision);
		RevWalk revWalk = new RevWalk(repo);
	    RevCommit commit = revWalk.parseCommit(lastCommitId);
	    RevTree tree = commit.getTree();

	    TreeWalk walk = new TreeWalk(repo);
	    walk.addTree(tree);
	    walk.setRecursive(true);

	    return walk;
	}

	public void deleteFolder(String folder) throws IOException{
		WindowCacheConfig cfg = new WindowCacheConfig();
	  cfg.setPackedGitMMAP(false);
	  cfg.install();

	  File t = new File(tempDir);
	  if (t.exists())
		FileUtils.delete(t, FileUtils.RECURSIVE | FileUtils.RETRY);
	}

	private void createDirIfDoesNotExist(String dir){
		File folder = new File(dir);
		folder.setReadable(true, false);
		folder.setExecutable(true, false);
		folder.setWritable(true, false);
		folder.mkdirs();
	}

	private void cloneRepo() throws Exception {
		deleteFolder(tempDir);
		createDirIfDoesNotExist(tempDir);

		Git.cloneRepository()
		.setURI(path)
		.setDirectory(new File(tempDir))
		.setCloneAllBranches(true)
		.call();
	}
```

```
	private String getBranchOfCommit(Repository repo, RevCommit commit)
throws Exception{
		ArrayList<String> branches = new ArrayList<String>();
		RevWalk walk = new RevWalk(repo);
		RevCommit target =
walk.parseCommit(repo.resolve(commit.getName()));

	for (Map.Entry<String, Ref> e : repo.getAllRefs().entrySet())
	{
		RevCommit c = walk.parseCommit(e.getValue().getObjectId());
	       if (e.getKey().startsWith(Constants.R_REMOTES))
		    if (walk.isMergedInto(target, c))

	branches.add(e.getValue().getName().replace(Constants.R_REMOTES, ""));
	}

		for (String branch : branches)
			if (branch.contains("master"))
				return branch;

		if (branches.size() > 0)
			return branches.get(0);

		return "";
	}

	/* PRIVATE METHODS - END */

}
```

Código C.6: SVNRepo

```
package api.vcs;

public class SVNRepo implements IRepository {
	private static String TRUNK_FOLDER = "trunk";
	private static String BRANCHES_FOLDER = "branches";
	private static String TAGS_FOLDER = "tags";
	private String path;
	private SVNClientManager manager;
	private ArrayList<SVNInfo> repoInfo;

	public SVNRepo(String path) throws Exception {

		this.path = path;
		this.manager = SVNClientManager.newInstance();
		this.repoInfo = new ArrayList<>();

		SVNURL.parseURIDecoded(getTrunkPath());

		initialize();
	}

	/* INTERFACE METHODS - BEGIN */

	@Override
```

```
	public ArrayList<Revision> getListOfRevisions() throws Exception {

		ArrayList<Revision> list = new ArrayList<Revision>();

		SVNURL url = SVNURL.parseURIEncoded(this.path);

		SVNRepository repository = SVNRepositoryFactory.create(url);

		Collection<?> logEntries = repository.log(new String[] {""} ,
null, 0, -1, true, true);

	  for (Iterator<?> entries = logEntries.iterator(); entries.hasNext();)
	  {
		SVNLogEntry logEntry = (SVNLogEntry) entries.next();

		if (logEntry.getAuthor() != null) {
			Revision rev = new Revision();

rev.setRevisionId(String.valueOf(logEntry.getRevision()));
			rev.setAuthor(logEntry.getAuthor());
			rev.setDate(logEntry.getDate());
			rev.setMessage(logEntry.getMessage());
			rev.setBranch(getBranch(logEntry));

			list.add(rev);
		}
	  }

	  Collections.sort(list, new RevisionComparator());

	  return list;
	}

	@Override
	public ArrayList<String> getListOfFiles(String revisionNumber) throws
Exception {
		getRevision(revisionNumber);

		ArrayList<String> list = new ArrayList<>();

		for (int i = 0; i < repoInfo.size(); i++)
			list.add(repoInfo.get(i).getURL().getPath());

		return list;
	}

	@Override
	public boolean downloadFile(String filePath, String revisionNumber,
String destFolder) throws Exception {
		getRevision(revisionNumber);
		return downloadFile(getInfo(filePath), destFolder);

	}

	@Override
	public boolean downloadAllFiles(String revisionNumber, String
destFolder) throws Exception {
		getRevision(revisionNumber);
```

```
            boolean allSucess = true;

            for (int i = 1; i < repoInfo.size(); i++)
            {
                  String srcPath =
repoInfo.get(i).getPath().replace(this.path, "");

                  SVNNodeKind nodeKind = repoInfo.get(i).getKind();

                  if (nodeKind == SVNNodeKind.FILE)
                  {
                        int index = srcPath.lastIndexOf("/");

                        String folders = index != -1 ? ((String)
srcPath.subSequence(0, index)) : "";

                        allSucess = !(downloadFile(repoInfo.get(i),
destFolder + "\\" + folders))
                                    ? false : allSucess;
                  }
                  else if (nodeKind == SVNNodeKind.DIR)
                  {
                        File folder = new File(destFolder + "\\" +
srcPath);
                        folder.setReadable(true, false);
                        folder.setExecutable(true, false);
                        folder.setWritable(true, false);
                        folder.mkdirs();
                  }
                  else allSucess = false;
            }

            return allSucess;
      }

      @Override
      public ArrayList<String> getListOfTags() throws Exception {

            final ArrayList<String> list = new ArrayList<>();

            try {
                  SVNURL svnurl = SVNURL.parseURIDecoded(getTagsPath());

              manager.getLogClient().doList(svnurl, SVNRevision.HEAD,
SVNRevision.HEAD, false, false,
                        new ISVNDirEntryHandler() {
                              public void handleDirEntry(SVNDirEntry
dirEntry) throws SVNException {
                                    if
(!dirEntry.getRelativePath().trim().equals(""))

      list.add(dirEntry.getRelativePath());
                        }
                  });
            }
            catch (SVNException e){
                  return list;
```

```
		}

		return list;
	}

	public ArrayList<String> getListOfBranches() throws Exception {
		final ArrayList<String> list = new ArrayList<>();
		list.add(TRUNK_FOLDER);

		try {
			SVNURL svnurl =
SVNURL.parseURIDecoded(getBranchesPath());
		  manager.getLogClient().doList(svnurl, SVNRevision.HEAD,
SVNRevision.HEAD, false, false,
				new ISVNDirEntryHandler() {
					public void handleDirEntry(SVNDirEntry
dirEntry) throws SVNException {
						if
(!dirEntry.getRelativePath().trim().equals(""))

	list.add(dirEntry.getRelativePath());
					}
				});
		}
		catch (SVNException e){
			return list;
		}

		return list;
	}

	@Override
	public String getURL() {
		return this.path;
	}

	/* INTERFACE METHODS - BEGIN */

	/* PRIVATE METHODS - BEGIN */

	private void initialize(){
		DAVRepositoryFactory.setup(); // http
		SVNRepositoryFactoryImpl.setup(); // svn
		FSRepositoryFactory.setup(); // file
	}

	private SVNInfo getInfo(String url)
	{
		SVNInfo info = null;
		int index = 0;

		while (info == null && index < repoInfo.size())
		{
			String u =
repoInfo.get(index).getURL().getPath().replace(this.path, "");

			if (u.equals(url))
				info = repoInfo.get(index);
```

```
                    index++;
             }

             return info;
      }

      private boolean downloadFile(SVNInfo info, String dest) throws
Exception {
             if (info != null)
             {
                    String filename = info.getPath().replace(getTrunkPath(),
"");
                    int index = filename.lastIndexOf("/");
                    filename = index != -1 ? filename.substring(index + 1,
filename.length()) : filename;

                    OutputStream output = new FileOutputStream(dest + "\\" +
filename);
                    manager.getWCClient().doGetFileContents(info.getURL(),
info.getRevision(), info.getRevision(), true, output);
                    return true;
             }

             return false;
      }

      private void getRevision(String rev) throws Exception {
             SVNRevision revision = SVNRevision.HEAD;
             long revNumber = Long.parseLong(rev);

             if (revNumber != 0)
                    revision = SVNRevision.create(revNumber);

             SVNURL svnurl = SVNURL.parseURIDecoded(this.path);
        manager.getWCClient().doInfo(svnurl, revision, revision,
SVNDepth.INFINITY, new SVNInfoHandler());
    }

      private String getBranch(SVNLogEntry logEntry) {
             if (logEntry.getChangedPaths().size() > 0) {
                    Object i =
logEntry.getChangedPaths().keySet().iterator().next();
                    SVNLogEntryPath entryPath = (SVNLogEntryPath)
logEntry.getChangedPaths().get(i);

                    if (entryPath.getPath().length() > 0) {
                           String p = entryPath.getPath().substring(1);


                           if (p.startsWith(BRANCHES_FOLDER)) {
                                  p = p.replace(BRANCHES_FOLDER, "");

                                  if (p.length() > 0){
                                         p = p.substring(1);
                                         int index = p.indexOf("/");

                                         return index == - 1 ? p :
p.substring(0, index);
```

```
                                }
                            }
                        }
                }

          return TRUNK_FOLDER;
        }

        private String getTagsPath() {
                return path + "/" + TAGS_FOLDER;
        }

        private String getTrunkPath() {
                return path + "/" + TRUNK_FOLDER;
        }

        private String getBranchesPath() {
                return path + "/" + BRANCHES_FOLDER;
        }

        private class SVNInfoHandler implements ISVNInfoHandler
        {
                @Override
                public void handleInfo(SVNInfo info) throws SVNException {
                        repoInfo.add(info);
                }

        }

        private class RevisionComparator implements Comparator<Revision> {

                @Override
                public int compare(Revision arg0, Revision arg1) {
                        return arg1.getDate().compareTo(arg0.getDate());
                }

        }

        /* PRIVATE METHODS - END */

}
```

APÊNDICE D - Resultados do estudo de caso

Esse apêndice é destinado apresentar outros resultados obtidos com o estudo de caso dos projetos de código aberto.

```
r318 | scottwilson | 2013-12-01 08:02:45 -0200 (Sun, 01 Dec 2013) | 1 line

Replaced String.isEmpty with String.length==0 for Android compatibility (thanks Wolfgang!)
------------------------------------------------------------------------
r317 | scottwilson | 2013-11-29 08:03:06 -0200 (Fri, 29 Nov 2013) | 1 line

Fixed issue with "&;" being incorrectly identified as a special character. Thanks to
Wolfgang for spotting this and providing a solution. See issue #98.
------------------------------------------------------------------------
r316 | scottwilson | 2013-11-29 08:01:40 -0200 (Fri, 29 Nov 2013) | 1 line

Added a test for invalid two-character "entity" - see bug #98
------------------------------------------------------------------------
r315 | scottwilson | 2013-11-28 13:43:05 -0200 (Thu, 28 Nov 2013) | 1 line

Added the HtmCleaner GUI sub-project - thanks to Marton Szeles for the contribution
------------------------------------------------------------------------
r314 | scottwilson | 2013-11-28 13:29:09 -0200 (Thu, 28 Nov 2013) | 1 line

Updated test to take account of adding newlines between CDATA markers
------------------------------------------------------------------------
r313 | scottwilson | 2013-11-28 13:28:36 -0200 (Thu, 28 Nov 2013) | 1 line

Broke out a problematic test from the XWiki set from Vincent into its own testcase to
make it easier to track
------------------------------------------------------------------------
r312 | scottwilson | 2013-11-28 13:24:08 -0200 (Thu, 28 Nov 2013) | 1 line

Updated test resources to take account of adding newlines between CDATA markers

                                    •••

r6 | vnikic | 2007-05-05 16:22:19 -0300 (Sat, 05 May 2007) | 1 line


------------------------------------------------------------------------
r5 | vnikic | 2007-05-05 06:35:45 -0300 (Sat, 05 May 2007) | 1 line

Escaping specil characters bug fixed.
------------------------------------------------------------------------
r4 | vnikic | 2007-05-04 16:35:09 -0300 (Fri, 04 May 2007) | 1 line

jdom.jar added to jars
------------------------------------------------------------------------
r3 | vnikic | 2007-05-04 14:04:36 -0300 (Fri, 04 May 2007) | 1 line


------------------------------------------------------------------------
r2 | vnikic | 2007-05-04 11:52:23 -0300 (Fri, 04 May 2007) | 7 lines

HtmlCleaner, version 1.2

* several bug fixes
* added option to omit HTML envelope in the resulting XML
* added option to treat unknown/deprecated tags as pure HTML content
* public accessor for root node added in HtmlCleaner class
* JDom serializer added
------------------------------------------------------------------------
r1 | vnikic | 2007-04-16 08:58:11 -0300 (Mon, 16 Apr 2007) | 1 line

Rejected commit: Default
```

Figura D.1: Lista de revisões com comando do SVN para o HtmlCleaner

```
Revisions:
ID: 318 | BY: scottwilson | AT: Sun Dec 01 08:02:45 BRST 2013 | IN: trunk
MESSAGE: Replaced String.isEmpty with String.length==0 for Android compatibility (thanks Wolfgang!)

ID: 317 | BY: scottwilson | AT: Fri Nov 29 08:03:06 BRST 2013 | IN: trunk
MESSAGE: Fixed issue with "&;" being incorrectly identified as a special character. Thanks to
Wolfgang for spotting this and providing a solution. See issue #98.

ID: 316 | BY: scottwilson | AT: Fri Nov 29 08:01:40 BRST 2013 | IN: trunk
MESSAGE: Added a test for invalid two-character "entity" - see bug #98

ID: 315 | BY: scottwilson | AT: Thu Nov 28 13:43:05 BRST 2013 | IN: trunk
MESSAGE: Added the HtmCleaner GUI sub-project - thanks to Marton Szeles for the contribution

ID: 314 | BY: scottwilson | AT: Thu Nov 28 13:29:09 BRST 2013 | IN: trunk
MESSAGE: Updated test to take account of adding newlines between CDATA markers

ID: 313 | BY: scottwilson | AT: Thu Nov 28 13:28:36 BRST 2013 | IN: trunk
MESSAGE: Broke out a problematic test from the XWiki set from Vincent into its own testcase to make
it easier to track

ID: 312 | BY: scottwilson | AT: Thu Nov 28 13:24:08 BRST 2013 | IN: trunk
MESSAGE: Updated test resources to take account of adding newlines between CDATA markers
                                    •••
ID: 6 | BY: vnikic | AT: Sat May 05 16:22:19 BRT 2007 | IN: trunk
MESSAGE:

ID: 5 | BY: vnikic | AT: Sat May 05 06:35:45 BRT 2007 | IN: trunk
MESSAGE: Escaping specil characters bug fixed.

ID: 4 | BY: vnikic | AT: Fri May 04 16:35:09 BRT 2007 | IN: trunk
MESSAGE: jdom.jar added to jars

ID: 3 | BY: vnikic | AT: Fri May 04 14:04:36 BRT 2007 | IN: trunk
MESSAGE:

ID: 2 | BY: vnikic | AT: Fri May 04 11:52:23 BRT 2007 | IN: trunk
MESSAGE: HtmlCleaner, version 1.2

* several bug fixes
* added option to omit HTML envelope in the resulting XML
* added option to treat unknown/deprecated tags as pure HTML content
* public accessor for root node added in HtmlCleaner class
* JDom serializer added

ID: 1 | BY: vnikic | AT: Mon Apr 16 08:58:11 BRT 2007 | IN: trunk
MESSAGE: Rejected commit: Default
```

Figura D.2: Lista de revisões com a API para o HtmlCleaner

```
15 deltalab             May 20  2013 ./
12 deltalab             May 20  2013 production/
15 deltalab             May 20  2013 www/
```

Figura D.3: Lista de ramos com comando do SVN para PHP QR Code

```
Branches:
trunk
production
www
```

Figura D.4: Lista de ramos com a API para PHP QR Code

```
23233 ezust               Oct 10 23:00 ./
23162 Vampire0            Aug 31 08:39 4.3.x/
23162 Vampire0            Aug 31 08:39 4.3.x-fix-view-leak/
23162 Vampire0            Aug 31 08:39 4.3.x-merge-request-2980833/
23162 Vampire0            Aug 31 08:39 4.4.x/
23162 Vampire0            Aug 31 08:39 4.4.x-merge-request-for-r18847-r18954-r19206-r19210/
23162 Vampire0            Aug 31 08:39 4.4.x-merge-request-for-r19197/
23162 Vampire0            Aug 31 08:39 4.4.x-merge-request-for-r19201/
23162 Vampire0            Aug 31 08:39 4.5.x/
23162 Vampire0            Aug 31 08:39 4.5.x-font-substitution-fix/
23162 Vampire0            Aug 31 08:39 5.0.x/
23162 Vampire0            Aug 31 08:39 5.0.x-2931321-disable-obsolete-plugins/
23233 ezust               Oct 10 23:00 5.1.x/
23162 Vampire0            Aug 31 08:39 ci_release_test/
23162 Vampire0            Aug 31 08:39 concurrency/
23162 Vampire0            Aug 31 08:39 jedit43_nostrings/
23162 Vampire0            Aug 31 08:39 lp_clone_test/
23162 Vampire0            Aug 31 08:39 new_bufferset_api/
23162 Vampire0            Aug 31 08:39 notifications/
23162 Vampire0            Aug 31 08:39 plugin_packages/
```

Figura D.5: Lista de ramos com comando do SVN para jEdit

```
Branches:
trunk
4.3.x
4.3.x-fix-view-leak
4.3.x-merge-request-2980833
4.4.x
4.4.x-merge-request-for-r18847-r18954-r19206-r19210
4.4.x-merge-request-for-r19197
4.4.x-merge-request-for-r19201
4.5.x
4.5.x-font-substitution-fix
5.0.x
5.0.x-2931321-disable-obsolete-plugins
5.1.x
ci_release_test
concurrency
jedit43_nostrings
lp_clone_test
new_bufferset_api
notifications
plugin_packages
```

Figura D.6: Lista de ramos com a API para jEdit

```
1276 thesmily            Nov 26  2012 ./
   2 thesmily            Sep 03  2004 Version 0.4.0/
   4 thesmily            Sep 03  2004 Version 0.5.0/
   8 thesmily            Sep 03  2004 Version 0.6.0/
  11 thesmily            Sep 03  2004 Version 0.7.0/
  13 thesmily            Sep 03  2004 Version 0.7.1/
  16 thesmily            Sep 03  2004 Version 0.8.0/
 104 thesmily            Nov 30  2005 Version 0.8.1 RC10/
  27 thesmily            Apr 14  2005 Version 0.8.1 RC6/
  38 thesmily            May 19  2005 Version 0.8.1 RC7/
  66 thesmily            Sep 24  2005 Version 0.8.1 RC9/
  80 thesmily            Oct 05  2005 Version 0.8.1 RC9 Patch 02/
  95 thesmily            Oct 12  2005 Version 0.8.1 RC9 Patch 03/
 123 thesmily            Jan 19  2006 Version 0.9.0/
 148 thesmily            Apr 21  2006 Version 0.9.1/
 225 thesmily            Jun 17  2006 Version 0.9.2/
 277 thesmily            Aug 25  2006 Version 0.9.3/
 350 thesmily            Dec 23  2007 Version 0.9.5/
 391 thesmily            Sep 18  2008 Version 0.9.6/
 447 thesmily            Feb 03  2009 Version 0.9.7/
 497 thesmily            Apr 08  2009 Version 0.9.8/
 595 thesmily            Jan 04  2010 Version 0.9.9/
 704 thesmily            May 28  2010 Version 1.0.0/
 726 thesmily            Jun 13  2010 Version 1.0.1/
 741 thesmily            Aug 16  2010 Version 1.0.2/
 801 thesmily            Nov 21  2010 Version 1.1.0/
 830 thesmily            Jan 23  2011 Version 1.2.0/
 872 thesmily            May 18  2011 Version 1.2.1/
 879 jahwe200            Jul 28  2011 Version 1.2.2/
 906 jahwe200            Jan 02  2012 Version 1.2.3/
 988 thesmily            Mar 11  2012 Version 1.3.0/
1039 thesmily            Mar 26  2012 Version 1.3.2/
1115 thesmily            Jun 04  2012 Version 1.4.0/
1132 thesmily            Jun 18  2012 Version 1.4.1/
1159 thesmily            Jul 05  2012 Version 1.4.2/
1180 thesmily            Jul 29  2012 Version 1.4.3/
1217 jahwe200            Aug 30  2012 Version 1.5.0/
1232 thesmily            Oct 24  2012 Version 1.5.1/
1276 thesmily            Nov 26  2012 Version 1.6.0/
```

Figura D.7: Lista de tags com comando do SVN para PDFCreator

```
Tags:
Version 0.4.0
Version 0.5.0
Version 0.6.0
Version 0.7.0
Version 0.7.1
Version 0.8.0
Version 0.8.1 RC10
Version 0.8.1 RC6
Version 0.8.1 RC7
Version 0.8.1 RC9
Version 0.8.1 RC9 Patch 02
Version 0.8.1 RC9 Patch 03
Version 0.9.0
Version 0.9.1
Version 0.9.2
Version 0.9.3
Version 0.9.5
Version 0.9.6
Version 0.9.7
Version 0.9.8
Version 0.9.9
Version 1.0.0
Version 1.0.1
Version 1.0.2
Version 1.1.0
Version 1.2.0
Version 1.2.1
Version 1.2.2
Version 1.2.3
Version 1.3.0
Version 1.3.2
Version 1.4.0
Version 1.4.1
Version 1.4.2
Version 1.4.3
Version 1.5.0
Version 1.5.1
Version 1.6.0
```

Figura D.8: Lista de tags com a API para PDFCreator

```
Rev: 1
 Directory of C:\Users\Paula\Desktop\Temp

12/03/2013  06:49 PM    <DIR>          .
12/03/2013  06:49 PM    <DIR>          ..
12/03/2013  06:49 PM    <DIR>          branches
12/03/2013  06:49 PM    <DIR>          tags
12/03/2013  06:49 PM    <DIR>          trunk
               0 File(s)              0 bytes

 Directory of C:\Users\Paula\Desktop\Temp\trunk

12/03/2013  06:49 PM    <DIR>          .
12/03/2013  06:49 PM    <DIR>          ..
12/03/2013  06:49 PM            19,248 filemove.avi
12/03/2013  06:49 PM             6,446 frmMain.frm
12/03/2013  06:49 PM             2,250 frmMain.frx
12/03/2013  06:49 PM            29,654 frmOptions.frm
12/03/2013  06:49 PM             3,051 frmOptions.frx
12/03/2013  06:49 PM             1,750 frmProcess.frm
12/03/2013  06:49 PM             2,250 frmProcess.frx
12/03/2013  06:49 PM             8,561 gsapi_vb6.bas
12/03/2013  06:49 PM             2,238 icon.ico
12/03/2013  06:49 PM             7,681 modGhostScript.bas
12/03/2013  06:49 PM             3,224 modLanguage.bas
12/03/2013  06:49 PM            12,909 modSave.bas
12/03/2013  06:49 PM             1,050 PDFCreator.vbp
12/03/2013  06:49 PM               299 PDFCreator.vbw
12/03/2013  06:49 PM               299 Projekt1.vbw
12/03/2013  06:49 PM               126 Source.txt
              16 File(s)        101,036 bytes
```

Figura D.9: Lista de arquivos com comando do SVN para PDFCreator

```
Files in revision 1:
/p/pdfcreator/code
/p/pdfcreator/code/trunk
/p/pdfcreator/code/trunk/gsapi_vb6.bas
/p/pdfcreator/code/trunk/modLanguage.bas
/p/pdfcreator/code/trunk/modSave.bas
/p/pdfcreator/code/trunk/frmProcess.frx
/p/pdfcreator/code/trunk/PDFCreator.vbp
/p/pdfcreator/code/trunk/frmOptions.frm
/p/pdfcreator/code/trunk/filemove.avi
/p/pdfcreator/code/trunk/frmMain.frm
/p/pdfcreator/code/trunk/PDFCreator.vbw
/p/pdfcreator/code/trunk/frmOptions.frx
/p/pdfcreator/code/trunk/Projekt1.vbw
/p/pdfcreator/code/trunk/Source.txt
/p/pdfcreator/code/trunk/frmMain.frx
/p/pdfcreator/code/trunk/modGhostScript.bas
/p/pdfcreator/code/trunk/icon.ico
/p/pdfcreator/code/trunk/frmProcess.frm
/p/pdfcreator/code/branches
/p/pdfcreator/code/tags
```

Figura D.10: Lista de arquivos com a API do SVN para

```
commit c182f3401c0f02feb0f9a8dc901702fb85853f94
Merge: a952272 1791f8d
Author: Artur Termenji <atermenji@gmail.com>
Date:   Mon Dec 2 04:25:10 2013 -0800

    Merge pull request #448 from pavlospt/master

    Fix clear_search_hostory in Greek Language

commit 1791f8d02021f45154fd74a4fa42576cbb63d8df
Author: Pavlos-Petros Tournaris <p.tournaris@gmail.com>
Date:   Thu Nov 28 04:59:04 2013 +0200

    Fix clear_search_hostory in Greek Language

    The translated string was missing the word "search" in Greek

commit a952272479da8bd514b7a81c568024d1f08fe5a6
Author: Artur Termenji <atermenji@gmail.com>
Date:   Sat Nov 23 12:59:30 2013 +0200

    Update readme. Now building requires Maven 3.1.1+

commit 5edf7d0fb8f52db570cca4eafcfe20f733e139a1
Merge: dc6f7ab 97615c7
Author: Artur Termenji <atermenji@gmail.com>
Date:   Sat Nov 23 02:57:52 2013 -0800

    Merge pull request #444 from crazymaster/patch-1

    Update Japanese translation

                                    •••

commit 54ffbb3b19c50656552da873de09917d139dec51
Author: Roberto Tyley <roberto@github.com>
Date:   Fri Oct 14 19:25:54 2011 +0100

    use com.github.mobile.android as we unambiguously own com.github.mobile ns

    The package id (which uniquely identifies the app on the Android Market) is
    now com.github.mobile.android rather than com.github.android.app -
    android.github.com is actually controlled by the AOSP.

    The package id is specified in the AndroidManifest.xml, and actual Java
    classes can be referred to relative to that package name, so it makes sense
    to change the Java package names to match.

    Also updated Maven POM's to use com.github.github as the group id - again,
    because we we completely control that namespace

commit 961dc214852e5baa1402263ee4c1ea0464873db4
Author: Roberto Tyley <roberto@github.com>
Date:   Fri Oct 14 18:32:30 2011 +0100

    creates an account, doesn't validate the credentials...

commit 7f0465a2728595ba47944a5682d07aa901d735be
Author: Roberto Tyley <roberto.tyley@gmail.com>
Date:   Wed Oct 12 23:36:58 2011 +0100

    initial shell GitHub app

    includes some accountauthenticator code, so will actually add an entry to
    accounts and sync, but actually grab your credentials and use them
```

Figura D.11: Lista de revisões com comando do Git para o GitHub Android App

```
Revisions:
ID: c182f3401c0f02feb0f9a8dc901702fb85853f94 | BY: Artur Termenji | AT: Mon Dec 02 10:25:10 BRST
2013 | IN: origin/master
MESSAGE: Merge pull request #448 from pavlospt/master

Fix clear_search_hostory in Greek Language

ID: 1791f8d02021f45154fd74a4fa42576cbb63d8df | BY: Pavlos-Petros Tournaris | AT: Thu Nov 28
00:59:04 BRST 2013 | IN: origin/master
MESSAGE: Fix clear_search_hostory in Greek Language

The translated string was missing the word "search" in Greek


ID: a952272479da8bd514b7a81c568024d1f08fe5a6 | BY: Artur Termenji | AT: Sat Nov 23 08:59:30 BRST
2013 | IN: origin/master
MESSAGE: Update readme. Now building requires Maven 3.1.1+


ID: 5edf7d0fb8f52db570cca4eafcfe20f733e139a1 | BY: Artur Termenji | AT: Sat Nov 23 08:57:52 BRST
2013 | IN: origin/master
MESSAGE: Merge pull request #444 from crazymaster/patch-1

Update Japanese translation
                                    •••

ID: 54ffbb3b19c50656552da873de09917d139dec51 | BY: Roberto Tyley | AT: Fri Oct 14 15:25:54 BRT
2011 | IN: origin/master
MESSAGE: use com.github.mobile.android as we unambiguously own com.github.mobile ns

The package id (which uniquely identifies the app on the Android Market) is
now com.github.mobile.android rather than com.github.android.app -
android.github.com is actually controlled by the AOSP.

The package id is specified in the AndroidManifest.xml, and actual Java
classes can be referred to relative to that package name, so it makes sense
to change the Java package names to match.

Also updated Maven POM's to use com.github.github as the group id - again,
because we we completely control that namespace


ID: 961dc214852e5baa1402263ee4c1ea0464873db4 | BY: Roberto Tyley | AT: Fri Oct 14 14:32:30 BRT
2011 | IN: origin/master
MESSAGE: creates an account, doesn't validate the credentials...


ID: 7f0465a2728595ba47944a5682d07aa901d735be | BY: Roberto Tyley | AT: Wed Oct 12 19:36:58 BRT
2011 | IN: origin/master
MESSAGE: initial shell GitHub app

includes some accountauthenticator code, so will actually add an entry to
accounts and sync, but actually grab your credentials and use them
```

Figura D.12: Lista de revisões com a API para o GitHub Android App

```
origin/3.0.x
origin/3.1.x
origin/3.2.x
origin/beanbuilder
origin/bootstrap
origin/cleanup-3.2.x
origin/conversation
origin/gh-pages
origin/master
origin/resource-handling
```

Figura D.13: Lista de ramos com comando do Git para Spring Framework

```
Branches:
origin/3.0.x
origin/3.1.x
origin/3.2.x
origin/beanbuilder
origin/bootstrap
origin/cleanup-3.2.x
origin/conversation
origin/gh-pages
origin/master
origin/resource-handling
```

Figura D.14: Lista de ramos com a API para Spring Framework

```
sdk-version-3.0
sdk-version-3.0.1
sdk-version-3.0.1.b
sdk-version-3.0.2
sdk-version-3.0.2.b
sdk-version-3.0.b
sdk-version-3.5
sdk-version-3.5.2
v1.0
v1.1
v1.2
v1.2.1
v1.2.2
```

Figura D.15: Lista de tags com comando do Git para Facebook SDK for Android

```
Tags:
sdk-version-3.0
sdk-version-3.0.1
sdk-version-3.0.1.b
sdk-version-3.0.2
sdk-version-3.0.2.b
sdk-version-3.0.b
sdk-version-3.5
sdk-version-3.5.2
v1.0
v1.1
v1.2
v1.2.1
v1.2.2
```

Figura D.16: Lista de tags com a API para Facebook SDK for Android

```
r1          r3          r42         r55
r10         r30         r43         r56
r11         r31         r44         r57
r12         r32         r45         r58
r13         r33         r46         r59
r14         r34         r47         r6
r15         r35         r48         r60
r16         r36         r49         r61
r17         r37         r5          r62
r18         r38         r50         r63
r2          r39         r51         r7
r25         r4          r52         r8
r28         r40         r53         r9
r29         r41/ROME    r54
```

Figura D.17: Lista de tags com comando do Git para Three.js

```
Tags:
r1          r3          r42         r55
r10         r30         r43         r56
r11         r31         r44         r57
r12         r32         r45         r58
r13         r33         r46         r59
r14         r34         r47         r6
r15         r35         r48         r60
r16         r36         r49         r61
r17         r37         r5          r62
r18         r38         r50         r63
r2          r39         r51         r7
r25         r4          r52         r8
r28         r40         r53         r9
r29         r41/ROME    r54
```

Figura D.18: Lista de tags com a API para Three.js

```
Rev: 84c5c3e5c301382ede5b1f04c7440ee9055589af
    Directory: C:\Users\Paula\Desktop\Homebrew
Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----          12/3/2013   6:30 PM            Cellar
d----          12/3/2013   6:31 PM            Formula
-a---          12/3/2013   6:30 PM         14 .gitignore
-a---          12/3/2013   6:31 PM       4354 README

    Directory: C:\Users\Paula\Desktop\Homebrew\Cellar
Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----          12/3/2013   6:31 PM            homebrew

    Directory: C:\Users\Paula\Desktop\Homebrew\Cellar\homebrew
Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---          12/3/2013   6:31 PM       1302 brew
-a---          12/3/2013   6:31 PM       3516 brewkit.rb
-a---          12/3/2013   6:30 PM       1801 unittest.rb

    Directory: C:\Users\Paula\Desktop\Homebrew\Formula
Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---          12/3/2013   6:30 PM        332 cmake.rb
-a---          12/3/2013   6:30 PM        742 dmd.rb
-a---          12/3/2013   6:30 PM        354 git.rb
-a---          12/3/2013   6:31 PM       2093 grc.rb
-a---          12/3/2013   6:30 PM        406 mad.rb
-a---          12/3/2013   6:30 PM        358 wget.rb
```

Figura D.19: Lista de arquivos com comando do Git para Homebrew

```
Files in revision 84c5c3e5c301382ede5b1f04c7440ee9055589af:
.gitignore
Cellar/homebrew/brew
Cellar/homebrew/brewkit.rb
Cellar/homebrew/unittest.rb
Formula/cmake.rb
Formula/dmd.rb
Formula/git.rb
Formula/grc.rb
Formula/mad.rb
Formula/wget.rb
README
```

Figura D.20: Lista de arquivos com a API para Homebrew