UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

DOUGLAS NASSIF ROMA JUNIOR

UMA FERRAMENTA PARA MINERAÇÃO DE DADOS DE PROJETOS DE SOFTWARE LIVRE E CRIAÇÃO DE REDES SÓCIO-TÉCNICAS

TRABALHO DE CONCLUSÃO DE CURSO

CAMPO MOURÃO
2013

DOUGLAS NASSIF ROMA JUNIOR

UMA FERRAMENTA PARA MINERAÇÃO DE DADOS DE PROJETOS DE SOFTWARE LIVRE E CRIAÇÃO DE REDES SÓCIO-TÉCNICAS

Trabalho de Conclusão de Curso de Graduação do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Me. Igor Scaliante Wiese

CAMPO MOURÃO
2013

ATA DA DEFESA DO TABALHO DE CONCLUSÃO DE CURSO

Às **dezenove horas** do dia **dois de maio de dois mil e treze** foi realizada no Mini-auditório do EAD da UTFPR-CM a sessão pública da defesa do Trabalho de Conclusão do Curso Superior de Tecnologia em Sistemas para Internet do acadêmico **Douglas Nassif Roma Junior** com o título **UMA FERRAMENTA PARA MINERAÇÃO DE DADOS DE PROJETOS DE SOFTWARE LIVRE E CRIAÇÃO DE REDES SÓCIO-TÉCNICAS**. Estavam presentes, além do acadêmico, os membros da banca examinadora composta pelo professor **Me. Igor Scaliante Wiese** (Orientador-Presidente), pelo professor **Me. Filipe Roseiro Cogo** e pelo professor **Dr. Marco Aurélio Graciotto Silva**. Inicialmente, o aluno fez a apresentação do seu trabalho, sendo, em seguida, arguido pela banca examinadora. Após as arguições, sem a presença do acadêmico, a banca examinadora o considerou **APROVADO** na disciplina de Trabalho de Conclusão de Curso e atribuiu, em consenso, a nota ____ (_____). Este resultado foi comunicado ao acadêmico e aos presentes na sessão pública. A banca examinadora também comunicou ao acadêmico que este resultado fica condicionado à entrega da versão final dentro dos padrões e da documentação exigida pela UTFPR ao professor Responsável do TCC no prazo de **onze dias**. Em seguida foi encerrada a sessão e, para constar, foi lavrada a presente Ata que segue assinada pelos membros da banca examinadora, após lida e considerada conforme.

Observações:

_____

_____

Campo Mourão, 02 de maio de 2013.

Prof. Me. Filipe Roseiro Cogo Membro	Prof. Dr. Marco Aurélio Graciotto Silva Membro	Prof. Me. Igor Scaliante Wiese Orientador

RESUMO

ROMA, Douglas N. Uma Ferramenta para Mineração de Dados de Projetos de Software, Criação de Redes e Cálculo de Métricas Sócio-Técnicas. 59 f. Trabalho de Conclusão de Curso – Curso Superior de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná – UTFPR. Campo Mourão, 2013.

Uma grande quantidade de dados – mensagens de *commit*, listas de e-mails, gerenciadores de tarefas (*issue*) e defeitos (*bug*) – é produzida durante o desenvolvimento de software. Diante deste cenário é importante a existência de ferramentas que auxiliem o entendimento da real situação do projeto. Para resolver esse problema existem ferramentas que auxiliam na mineração dos dados de forma eficiente. Mas estas ferramentas são inflexíveis e criadas para agir em domínios específicos. Neste trabalho foi desenvolvida uma arquitetura que possibilita a mineração de dados de projetos de software hospedados no `GitHub`, geração de redes e cálculo de métricas sociais, técnicas e sócio-técnicas. Deste modo, a ferramenta abre espaço para que pesquisadores e donos de projetos acrescentem extensões que solucionem seus próprios problemas, tais como: oferecer uma plataforma para pesquisadores implementarem suas próprias redes e métricas; facilitar o entendimento da real situação de um projeto de desenvolvimento de software; e prover a descoberta de tendências em um projeto de software.

Palavras-chaves: Mineração de dados. Engenharia de software. Rede Social. Métrica.

ABSTRACT

ROMA, Douglas N. An Tool for Data Mining Projects Software, Networking and Calculation Metrics Socio-Technical. 59 f. Trabalho de Conclusão de Curso – Curso Superior de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná – UTFPR. Campo Mourão, 2013.

A large amount of data – commit messages, mailing lists, issue and bug managers – is produced during the software development. In this scenario it is important to have tools that help the understanding of real project status. To solve this problem there are tools that assist in mining the data efficiently. But these tools are inflexible and created to act in specific fields. In this work we developed an architecture that allows data collection software project hosted on `GitHub`, network generation and calculation of metrics social, technical and socio-technical. Thus, the tool leaves room for researchers and project owners add extensions that solve their own problems, such as: providing a platform for researchers to implement their own networks and metrics; facilitate the understanding of the real situation of a software development project; and provide for the discovery of trends in a software project.

Keywords: Data Mining. Software engineering. Social Networking. Metric.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Uma grande quantidade de dados é produzida durante o desenvolvimento de software. Desenvolvedores utilizam diversas ferramentas para coordenar suas tarefas, trabalhar cooperativamente sobre artefatos e comunicar informações sobre o projeto. Controladores de versão (SVN[1], GIT[2], entre outros), gerenciadores de *issues*[3] e defeitos (Jira[4], Bugzilla[5]) e listas de e-mails (Google Groups[6], Yahoo Grupos[7]), são fontes de dados que podem ser mineradas e interpretadas (HASSAN *et al.*, 2008). Alguns ambientes, como o GitHub[8] e o SourceForge[9], oferecem a integração de diferentes serviços para o gerenciamento de projetos. Entre esses serviços estão o armazenamento do código fonte em diferentes repositórios, listas de discussão e gerenciamento de *issues*.

Apesar do grande volume de dados e dos tipos de métricas que tentam auxiliar na descoberta de indicadores, projetos de software continuam a ser difíceis de gerenciar. A mineração de dados na engenharia de software tem emergido como uma evidente área de pesquisa, oferecendo formas de interpretar a quantidade abundante de dados e, consequentemente, auxiliar na resolução de diversos problemas que envolvem o desenvolvimento de software (XIE; HASSAN, 2009).

Diante desse cenário, construir ferramentas que possibilitem a interpretação de dados provenientes do processo de Engenharia de Software é importante para aumentar o entendimento do desenvolvimento de software, dos seus processos e artefatos resultantes (HASSAN; XIE, 2010). Analisar dados provenientes dos repositórios de software pode auxiliar gestores na alocação adequada de recursos (BUSE; ZIMMERMANN, 2010).

Nesse contexto, é possível observar interações reais de desenvolvimento

[1] http://subversion.apache.org
[2] http://www.git-scm.com
[3] Tarefas a serem desenvolvidas em um projeto de software como a correção de um problema ou uma nova funcionalidade.
[4] http://www.atlassian.com/software/jira
[5] http://www.bugzilla.org
[6] http://groups.google.com/
[7] http://br.groups.yahoo.com/
[8] http://www.github.com
[9] http://www.sourceforge.com

para criar modelos que possibilitem uma melhor previsão, planejamento e compreensão dos aspectos técnicos e sociais que ocorrem em um projeto de software. Uma vez que os dados tenham sido minerados, diversas análises podem ser feitas, tais como: auxiliar desenvolvedores a encontrar informações (recomendação), estudar aspectos sociais, realizar predições de defeitos, estudar como as comunidades de software livre se organizam, replicações de estudos empíricos, entre outras áreas (GODFREY *et al.*, 2009).

O objetivo deste trabalho é construir uma ferramenta para coleta de dados, geração de redes e cálculo de métricas para os repositórios de software hospedados no `GitHub`, com isso espera-se: oferecer uma plataforma para pesquisadores implementarem suas próprias redes e métricas; facilitar o entendimento da real situação de um projeto de desenvolvimento de software; e prover a descoberta de tendências em um projeto de software.

A ferramenta utilizará uma `API` (*Application Programming Interface* ou Interface de Programação de Aplicativos) de comunicação via `JSON` disponibilizada pelo próprio `GitHub` para obtenção dos dados do site e desenvolverá outros dois módulos: um para geração e exportação das redes construídas a partir dos dados minerados; e outro para cálculo e exportação de métricas sociais, técnicas e sócio-técnicas.

Diversas ferramentas foram encontradas na literatura e inspiram os requisitos para a construção deste trabalho, mas nenhuma delas oferece a possibilidade de coletar dados do `GitHub`.

O restante desse trabalho esta estruturado da maneira apresentada a seguir. O Capítulo 2 apresenta o referencial teórico e a revisão bibliográfica; o Capítulo 3 apresenta a arquitetura e o funcionamento da ferramenta desenvolvida, bem como, um exemplo de implementação de métricas e redes utilizando a infraestrutura proposta; por fim o Capítulo 4 apresenta a conclusão.

2 REFERENCIAL TEÓRICO

Neste capítulo serão abordados conceitos sobre fontes de dados, mineração de dados na engenharia de software e a comparação entre os trabalhos relacionados.

2.1 MINERAÇÃO DE DADOS DE ENGENHARIA DE SOFTWARE E ASPECTOS SÓCIO-TÉCNICOS

A mineração de dados na engenharia de software pode auxiliar no entendimento das atividades do processo de desenvolvimento, bem como permitir a construção de conhecimento para planejar, entender e predizer aspectos relacionados a um projeto (XIE; HASSAN, 2011).

Esta seção apresenta conceitos sobre as diferentes fontes de dados da engenharia de software, a definição de dados sócio-técnicos, o repositório `GitHub` e suas funcionalidades, o `Weka`[10] (HALL *et al.*, 2009) que é um ambiente para análise de dados utilizando algoritmos de aprendizagem de máquina, o `PAJEK`[11] que é uma ferramenta que permite a visualização e cálculo de redes, e o `Gephi` que faz visualização e manipulação de grafos e redes.

2.1.1 FONTES DE DADOS E ASPECTOS SÓCIO-TÉCNICOS

Os dados provenientes do desenvolvimento de software podem ser obtidos de diferentes fontes. Essas fontes de dados normalmente são serviços que podem ser utilizados por repositórios de projetos. O Quadro 1 apresenta as

[10] **http://www.cs.waikato.ac.nz/ml/weka/**
[11] **http://pajek.imfm.si**

diferentes fontes que podem fornecer dados para análise de projetos de software.

Fontes de Dados	Descrição
Controle de Versão	Armazenam um histórico do projeto. Eles rastreiam todas as mudanças que ocorrem em um código-fonte ou qualquer artefato, associando meta dados a cada mudança. Os meta dados normalmente são: desenvolvedor que fez a mudança, data e hora da mudança, mensagem que descreve a mudança ocorrida e uma lista de artefatos modificados. Exemplos de controle de versão são: `Git`, `Mercurial`[12], `Bazaar`[13], `CVS`[14] e `Subversion`.
Sistemas de gerenciamento de defeitos e/ou *issues*	São repositórios que rastreiam a resolução de defeitos e/ou novos requisitos dos projetos. Informações sobre o relato do defeito ou a solicitação de novas funcionalidades são armazenadas. Tarefas são criadas e associadas a desenvolvedores. `Bugzilla` e `Jira` são exemplos de ferramentas para gerenciamento de *issues.*
Logs (processo de registro de eventos em um sistema)	Diferentes ferramentas utilizadas para o desenvolvimento de aplicações podem gerar *logs* com informações de compilações, execuções do software, utilização de arcabouços de testes, cliques em funcionalidades específicas de IDE's (Ambiente de Desenvolvimento Integrado), entre outros.
Históricos de Comunicação	Diversas ferramentas podem ser utilizadas para permitir a comunicação e o armazenamento do histórico de mensagens trocadas entre os desenvolvedores. Listas de e-mails, listas de discussões, `IRC` (*Internet Relay Chat*), bate-papo, são exemplos destes mecanismos.
Repositórios de Projeto de Software	São sites que oferecem o conjunto de serviços descritos acima para muitos projetos. `Sourceforge`, `Google Code`[15] e o `GitHub` são exemplos destes repositórios.

Quadro 1 – Fontes de dados para Mineração de dados de Engenharia de Software.

Fonte: Adaptado de Hassan e Xie (2010).

Segundo Hassan e Xie (2010), é possível realizar diferentes análises com os dados coletados e aplicar diferentes algoritmos de mineração, para explicar diferentes tarefas do processo de desenvolvimento do software. O Quadro 2 apresenta a sugestão de mineração e aplicação de algoritmos dos autores.

Dados	Algoritmos de mineração	Tarefas da engenharia de software
Sequência: execução estática de código, *co-changes*[16], etc.	Regras de associação, frequência de itens, clusterização, classificação, etc.	Programação, manutenção, detecção de falhas, depuração, etc.

[12] http://mercurial.selenic.com
[13] http://bazaar.canonical.com
[14] http://cvs.nongnu.org/
[15] http://code.google.com
[16] Alteração paralela e cooperativa de código-fonte.

Grafos: chamadas dinâmicas e estáticas, grafos de dependência, etc.	Grafos, clusterização, classificação, etc.	Detecção de grafos, depuração, etc.
Texto: relatório de falhas, e-mail, comentário de código, documentação.	Classificação, clusterização, análise textual.	Manutenção, detecção de falhas, depuração.

Quadro 2 – Dados de engenharia de software, exemplos de algoritmos e tarefas do processo de software para as análises.

Fonte: Adaptado de Hassan e Xie (2010).

As fontes de dados citadas podem fornecer dados de diferentes aspectos. Por exemplo, uma mensagem entre dois desenvolvedores em uma lista de e-mails fornece informações da interação social do desenvolvimento, indicando com quem um desenvolvedor interage socialmente. Se observarmos códigos-fonte submetidos ao repositório, podemos obter informações técnicas a respeito dos artefatos, como a quantidade de linhas modificadas. No entanto se olhar para uma mensagem de e-mail entre dois desenvolvedores anexado a um artefato, obtêm-se informações sócio-técnicas, indicando quais desenvolvedores influenciam cooperativamente na alteração do código-fonte.

Uma vez que a natureza do desenvolvimento do software é colaborativa, entende-se que a cooperação entre as pessoas para realização das tarefas permite a realização de análises a partir de aspectos sociais e técnicos. Repositórios de código-fonte, listas de e-mail e sistemas de gerenciamento de *issues* contêm dados que permitem a reconstrução da visão do projeto ao longo do tempo e, desta forma, permitem análises de relacionamentos sócio-técnicos (BIRD *et al.*, 2011).

Uma das formas de analisar os relacionamentos sócio-técnicos é por meio de análises de redes sociais, uma vez que uma rede de artefatos técnicos pode ser obtida de códigos minerados do repositório e informações sociais podem ser obtidas sobre as relações entre as pessoas, sejam elas a colaboração durante a realização de uma tarefa ou a comunicação existente entre os desenvolvedores (SARMA *et al.*, 2009).

A mineração dos dados também pode ser utilizada para relacionar os acontecimentos a aspectos de evolução, considerando a temporalidade em que os fatos acontecem. Por exemplo, o controle de versão, os gerenciadores de *issues* e os históricos de comunicação oferecem informações sobre o progresso dos projetos.

Já os *logs* oferecem informações sobre o contexto e do tempo real dos acontecimentos.

A temporalidade é importante, pois estabelece formas diferentes de análises. É importante relacionar diferentes fontes de dados, questionamentos e *insights*[17] (ou análises) que podem ser interpretados a partir dos dados minerados.

A Figura 1 apresenta a visão de Buse e Zimmermann (2010). Eles relacionam informações, temporalidade e *insights* para analisar projetos de software. A temporalidade envolve dados que são obtidos no passado, no presente e que podem gerar dados no futuro. Os *insights* relacionam-se com as possibilidades apresentadas por Godfrey (*et al.*, 2009), que destacam a importância da recomendação e predições como tópicos de pesquisa emergentes.

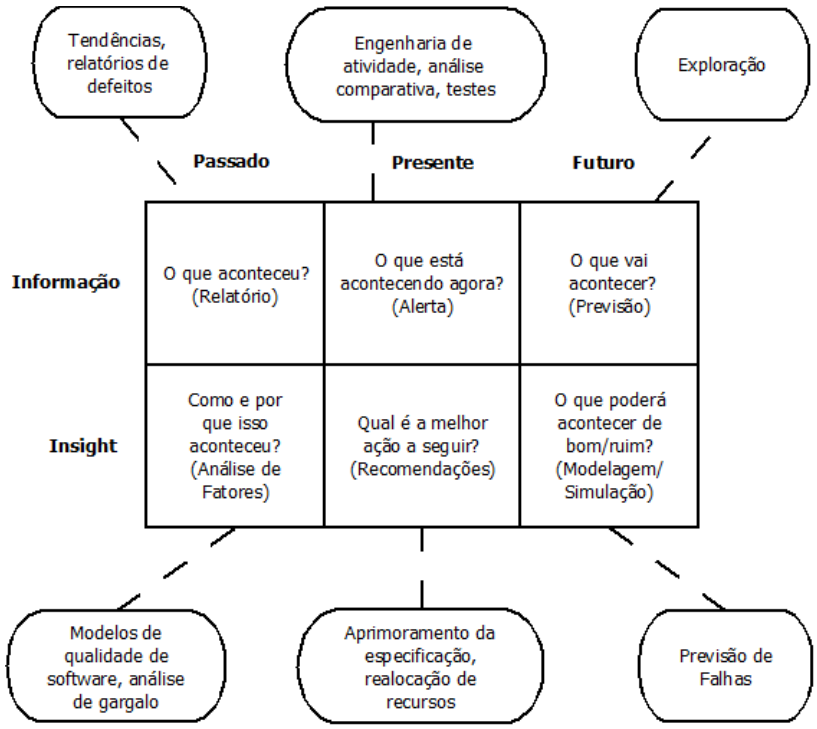

Figura 1 – Informações e *Insights*.

Fonte: Adaptado de (BUSE; ZIMMERMANN, 2010)

A Figura 2 apresenta uma abordagem para compor tipos de análises para obter diferentes resultados. É possível coletar diferentes meta dados, considerando diferentes fontes e instantes do tempo; combinar métricas para obter uma

[17] **Percepções obtidas através da analises dos dados.**

quantificação de aspectos distintos do desenvolvimento do software, e posteriormente aplicar modelos e simulações, tais como, algoritmos de predição e agrupamento (*clustering*), ou estudos qualitativos para explicar determinadas observações realizadas.

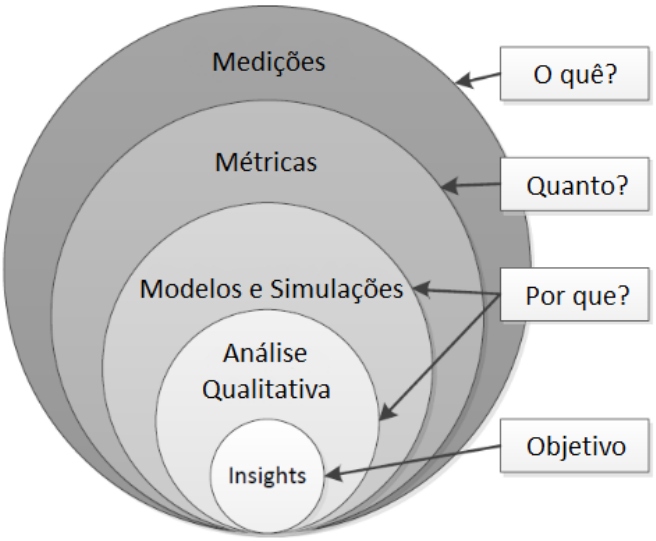

Figura 2 – A regra de análise para compor muitos tipos de análises e formular *insights* mais complexos.

Fonte: Adaptado de Buse e Zimmermann (2010)

2.2 `GitHub` e suas funcionalidades

As aplicações sociais na *web* permitem aos usuários monitorar e acompanhar suas atividades e de outros desenvolvedores que participam de um mesmo projeto, independentemente da sua localização. Pessoas participam de um conjunto rico de interações sociais. Podem descobrir objetivos técnicos, visualizar quando o código é editado e descobrir quais usuários participam de cada modificação. Usuários combinam essas inferências em efetivas estratégias para a coordenação do trabalho, avançando habilidades técnicas e gestão da sua reputação (DABBISH *et al.,* 2012).

O `GitHub` é uma aplicação *web* que fornece um conjunto de serviços voltados a "codificação social" e utilizam o controle de versão distribuído chamado `Git`. Entre as funcionalidades do `GitHub` destacam-se aquelas voltadas aos

desenvolvedores, artefatos e ações. A Figura 3 apresenta a página de perfil de um desenvolvedor no `GitHub`. Nos perfis, é possível obter informações sobre dados pessoais dos desenvolvedores, como por exemplo, sua localização (1), dados sobre os projetos que ele participa (2), repositórios que ele acompanha (*watching*) (3), as pessoas que ele acompanha (*following*) (3), por quem é acompanhado (*followers*) (3) e as suas últimas atividades nos projetos que ele participa (*Public Activity*) (5).

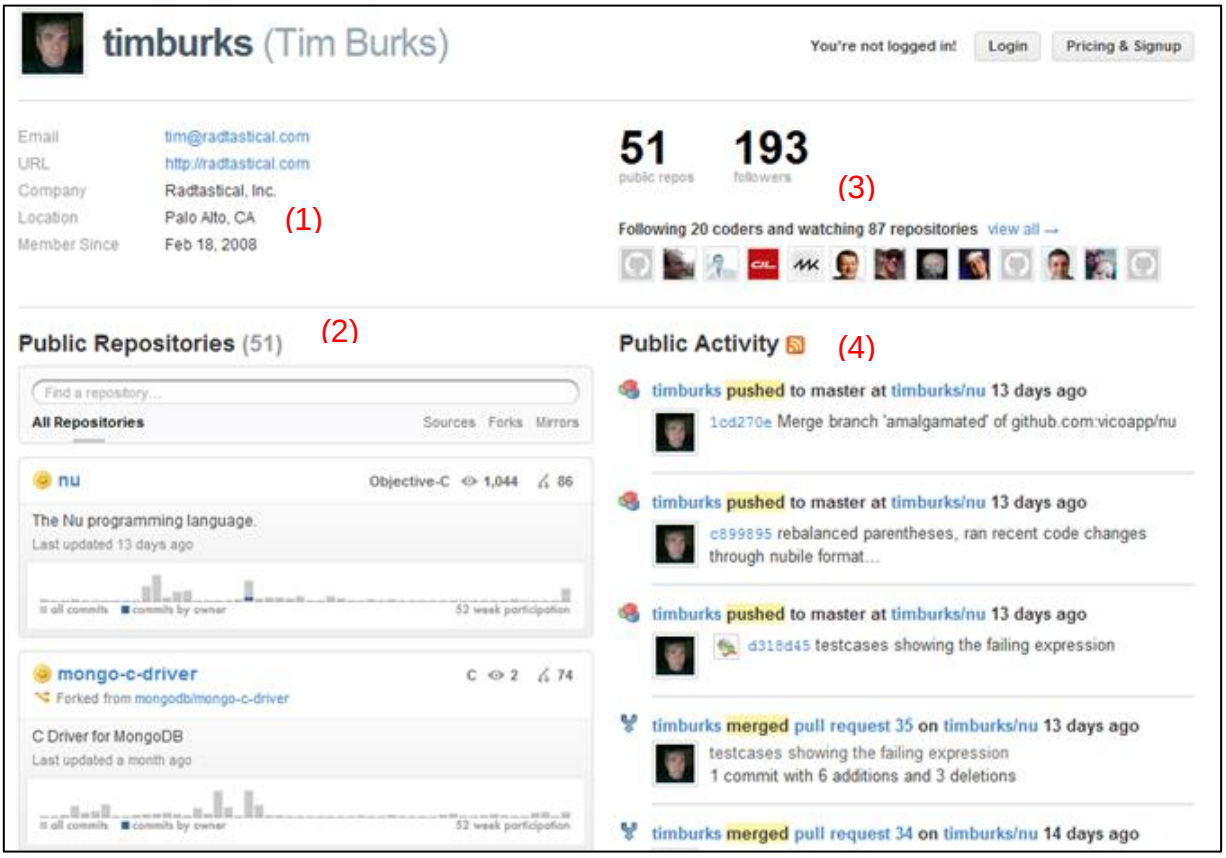

Figura 3 – Exemplo do perfil de um desenvolvedor no `GitHub`.

Fonte: www.github.com

Uma *issue* representa uma tarefa que precisa ser executada em um projeto, tal como a correção de um defeito ou a implementação de uma nova funcionalidade no software. No `GitHub`, desenvolvedores podem tanto comentar uma *issue* como também enviar soluções (também chamados de *patch* ou *pull requests*).

Um *pull request* significa o envio do código-fonte que implementa aquela *issue*. A Figura 4 apresenta a interação de desenvolvedores em uma *issue*. Observe que vários participantes podem comentar em uma *issue*, ou até mesmo comentar em trechos do código que foram enviados para a sua implementação.

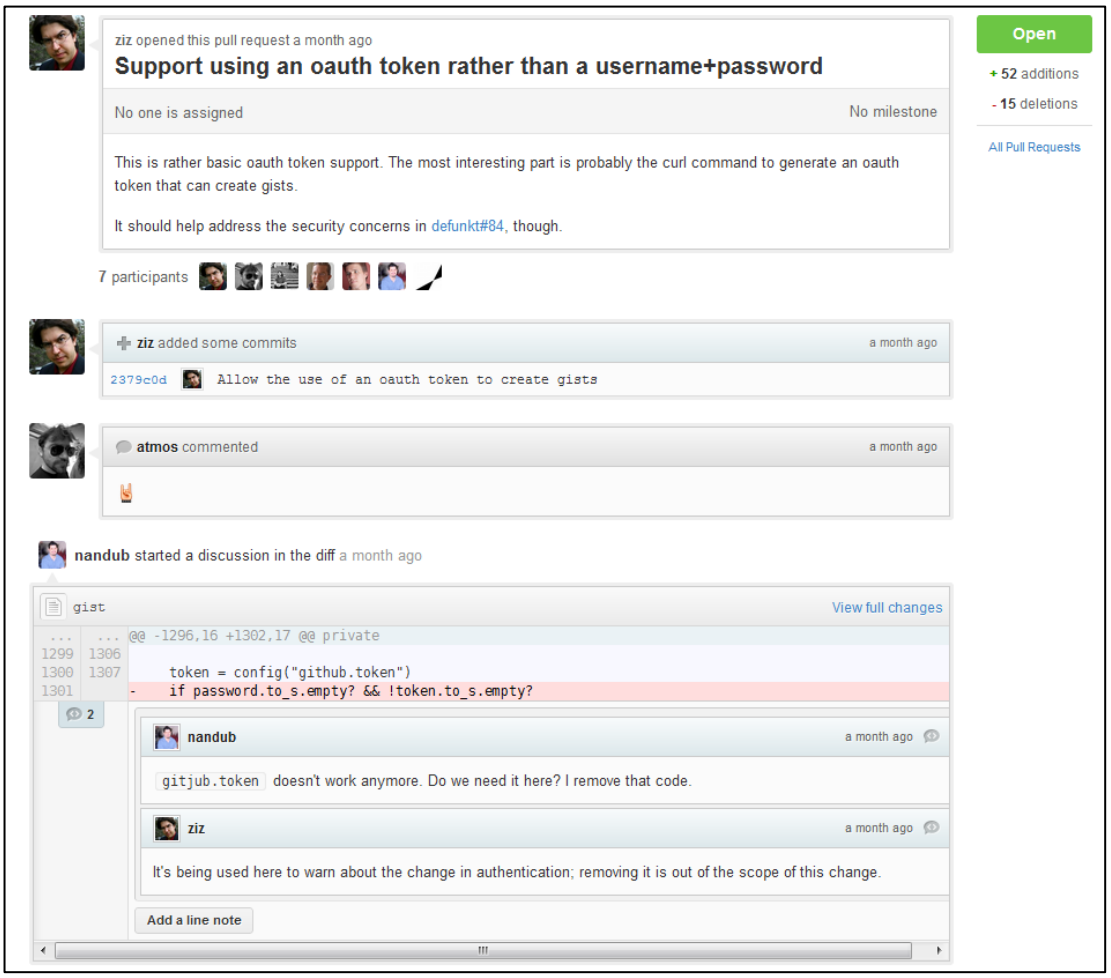

Figura 4 – Exemplo de uma *issue* em um projeto.

Fonte: www.github.com

Diferentemente dos repositórios centralizados, o `GitHub`, por utilizar um repositório distribuído, possibilita que desenvolvedores mantenedores de um projeto, também chamados de *commiters,* interajam com desenvolvedores que contribuem com um projeto. Estes colaboradores submetem seus códigos que ficam pendentes na opção de *pull request*. Um *commiter* pode avaliar a submissão, aceitando-a ou rejeitando-a.

A Figura 5 mostra uma lista de submissões feitas por colaboradores que aguardam aprovação. Enquanto aguardam, outros desenvolvedores e colaboradores podem interagir com o código, comentando e adicionando informações sobre a solução.

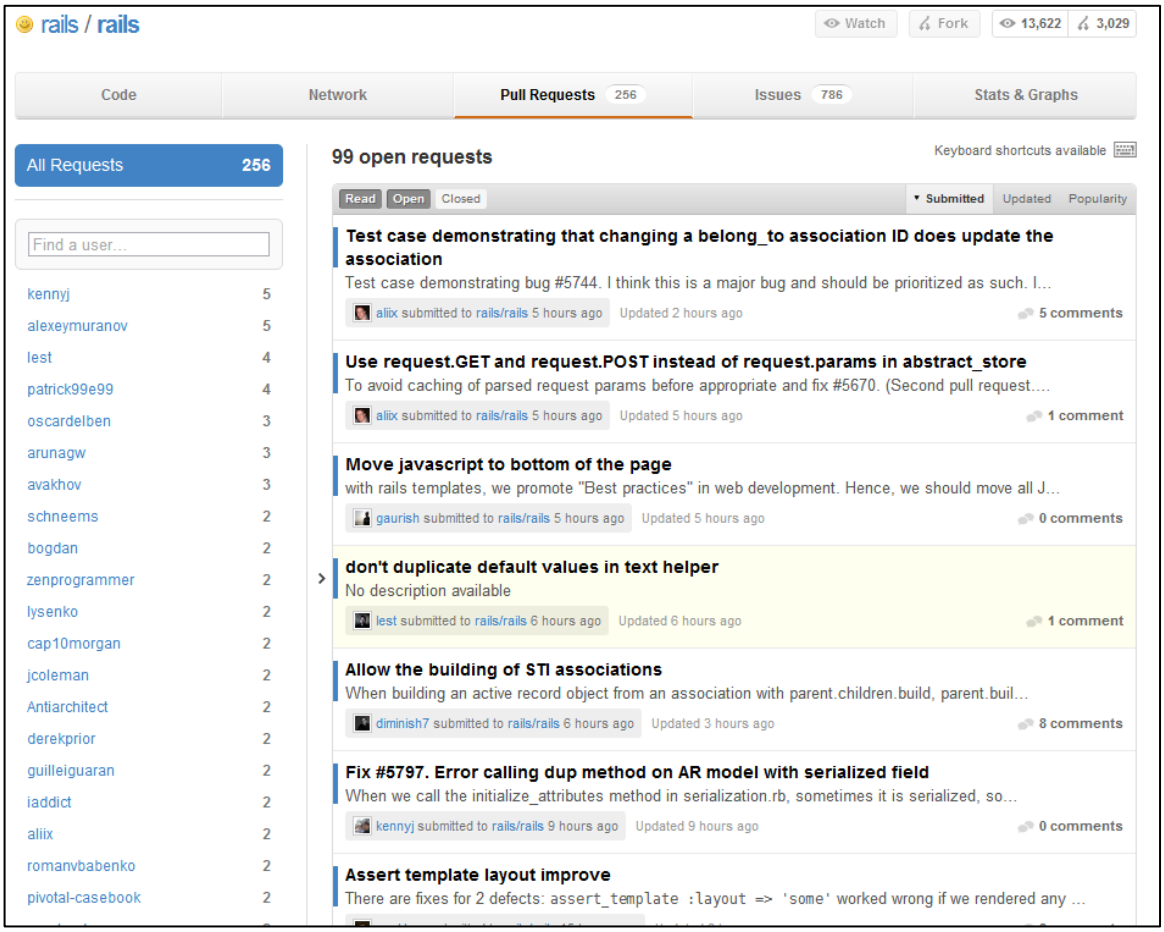

Figura 5 – Exemplo de *pull requests* de um projeto

Fonte: www.github.com.

A obtenção de dados do GitHub permite os pesquisadores analisar informações técnicas e sociais a respeito do código, tornando-se o grande diferencial para os repositórios centralizados e sites de hospedagem de projetos de software como o SourceForge.

2.3 ANÁLISE E VISUALIZAÇÃO DE REDES

Com o objetivo de agregar valores as funcionalidades deste trabalho, foram criadas formas para permitir a exportação das redes e métricas geradas em arquivos com formato CSV. Com isso é possível importá-los em outras ferramentas e utilizar funções que este trabalho não oferece. Para isso foram selecionadas três ferramentas conceituadas na área de análise, visualização e manipulação de redes,

grafos e matrizes. São elas o Pajek, Weka e Gephi apresentados nesta Seção.

2.3.1 Pajek e suas funcionalidades

O Pajek é uma ferramenta criada para gerar visualizações de redes complexas (NOOY *et al.*, 2004). O programa começou a ser desenvolvido em 1996 na *University of Ljubljana* (Eslovênia). Muito utilizado no meio acadêmico, possui uma ampla aplicação: é utilizado para geração de redes de comunicação, redes de disseminação de doenças, redes na Internet, mineração de dados, análise de ligação entre moléculas químicas, além das redes de colaboração, de citação, genealógicas, etc.

Este software permite identificar agrupamentos que formam subgrupos dentro das redes e identificar separadamente vértices pertencentes aos mesmos agrupamentos. Desse modo, o programa decompõe as redes muito complexas em redes menores sem desvinculá-las do contexto global da rede maior.

Há três formas de se desenhar uma rede com o Pajek. A primeira é realizada através da criação de matriz no próprio programa. A segunda possibilidade é a geração de redes a partir de processadores de texto. Por fim, também é possível a importação de dados de outros software como: UCINET DL, genealogical GED, e alguns formatos moleculares: BS (*Ball and Stick*), MAC (*Mac Molecule*) e MOL (MDL MOLfile).

Um dos objetivos neste trabalho é criar a possibilidade de exportação das redes para utilizar o Pajek para realizar a visualização das redes e grafos por meio da importação de arquivos no formato CSV, esses arquivos são criados e exportados no módulo de cálculo de métricas pela ferramenta desenvolvida neste trabalho.

2.3.2 Weka e suas funcionalidades

O Weka é uma ferramenta desenvolvida na Universidade de Waikato

(Nova Zelândia) e que começou tomar sua forma moderna em 1997. Essa ferramenta está sob a licença GNU *General Public License* (GPL), é escrita na linguagem `Java` e contém uma GUI (Interface Gráfica do Usuário) para interagir com arquivos de dados e produzir resultados visuais (tabelas e curvas).

Ela também possui uma API que permite sua incorporação aos seus próprios aplicativos. Com isso podemos executar tarefas como mineração e classificação de dados com algoritmos *bayesianos* e árvore de decisão.

Um dos objetivos neste trabalho é criar a possibilidade de exportação das métricas para utilizar o `WEKA` para executar algoritmos de aprendizagem de máquina por meio da importação de arquivos no formato `CSV`, esses arquivos são criados no módulo de cálculo de métricas pela ferramenta desenvolvida neste trabalho.

2.3.3 `Gephi` e suas funcionalidades

O `Gephi`[18] (BASTIAN *et al.,* 2009) é um software de código aberto para visualização, exploração e manipulação de gráficos e redes. Suas principais funcionalidades são:

- visualização: possui um rápido mecanismo de visualização de gráficos que acelera a compreensão e a descoberta de padrões em grande escala.
- aceita múltiplos formatos de arquivos: é capaz de importar a exportar múltiplos formatos de arquivos como CSV, NET (`Pajek`), GML, DOT, DL, `Ucinet`, `Netdraw`, entre outros.
- *plug-ins*: baseado na plataforma `NetBeans`[19], ele conta com uma gama de *plug-ins* independentes que acrescentam funcionalidades.

Um dos objetivos neste trabalho é criar a possibilidade de exportação das redes para utilizar o `Gephi` para visualização e exploração por meio da importação de arquivos no formato `CSV`, esses arquivos são criados no módulo de geração de redes pela ferramenta desenvolvida neste trabalho.

[18] http://gephi.org/
[19] http://www.netbeans.org

2.4 REVISÃO BIBLIOGRÁFICA

Neste capítulo são apresentados trabalhos relacionados considerados importantes na análise de dados de repositórios de software, e suas relevâncias para estes trabalhos estão destacadas no Quadro 3 da Seção 2.5.

2.4.1 xFlow

A xFlow (SANTANA *et al.*, 2011) foi criada com base em funções existentes na TransFlow (COSTA; *et al.,* 2009) e com algumas ideias pontuais encontradas também na OSSNetwork (BALIEIRO *et al.*, 2008).

A OSSNetwork coleta dados de fóruns e listas de e-mails possibilitando a construção de redes sociais, posteriormente oferece visualização das redes em forma de grafos com filtros que facilitam a análise dando uma noção de métricas na sua manipulação, permitindo a exclusão de nós com base em distância, centralidade e intermediação.

A TransFlow por sua vez, obtém informações a partir de *logs* de controles de versões. Ela não é uma ferramenta de análise social, mas consegue gerar uma visualização de rede sócio-técnica com base nas contribuições das pessoas junto ao projeto. Não possui cálculos de métricas para os atores, limitando-se apenas para artefatos.

Finalmente, a xFlow, possibilita o estudo sobre como os sistemas de software evoluem ao longo do tempo e como os desenvolvedores participam desse processo (SANTANA *et al.*, 2011). Dentre suas principais características, destacamos:

- coleta de dados de CVS, com conectores para repositórios Subversion e Git; listas de e-mails, fóruns de discussão, etc.;

- análise de dependências estáticas por meio de *call graph*, lógicas por meio de análises em modificações paralelas de artefatos e requisitos de coordenação utilizando multiplicação de matrizes;

- métricas de projeto como Densidade e Coeficiente de Cluster, métricas de artefatos baseadas em Linhas de Códigos, Grau de Centralidade e Grau de Intermediação, e também métricas de contribuição baseando-se em artefatos adicionados, removidos e modificados;

- visualizações em linha *(Figura 6(a))*, dispersão e atividades *(Figura 6 (b))*, grafos *(Figura 6 (c))*, *treemap (Figura 6(d))* e gráficos de barras e área empilhada *(Figura 6 (e))*.

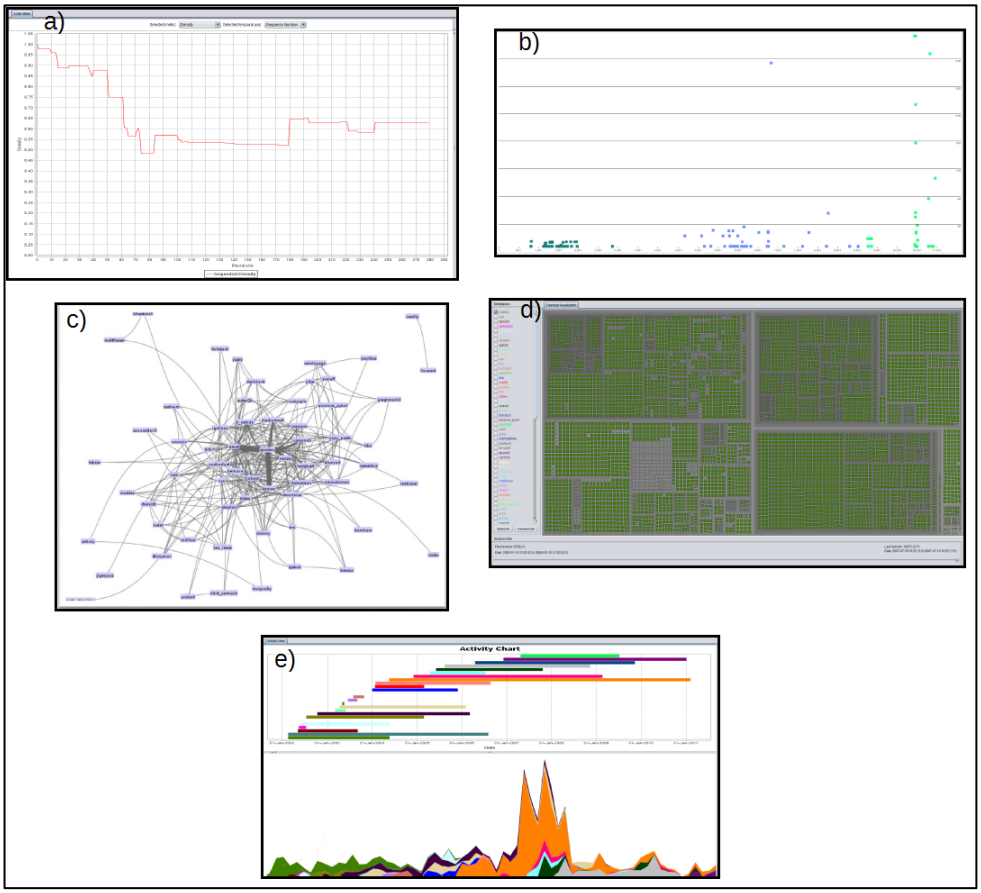

Figura 6 – `xFlow` - Telas de visualizações, grafos e outros.

Fonte: Adaptado de Santana (*et al.*, 2011).

2.4.2 `EvolTrack-SocialNetwork`

A `EvolTrack-SocialNetwork` (MAGDALENO *et al.*, 2010) foi

implementada como um *plug-in* para o Eclipse[20] sendo uma extensão da ferramenta `EvolTrack` (CEPEDA *et al.*, 2010).

A ferramenta permite capturar e visualizar o ciclo de evoluções de um projeto de software. Dentre as funções da `Evoltrack`, destacam-se: (i) a captura de informações referentes ao histórico da evolução de um projeto de software a partir de uma fonte de dados, tais como ferramentas de controle de versão (`Subversion`, `CVS`, etc.) ou ambientes de desenvolvimento de software (`Eclipse` ou `NetBeans`); (ii) as transformações/marcações sobre informações coletadas de acordo com o objetivo da análise; (iii) as opções de visualização das informações de evolução do software utilizando alguma forma de representação.

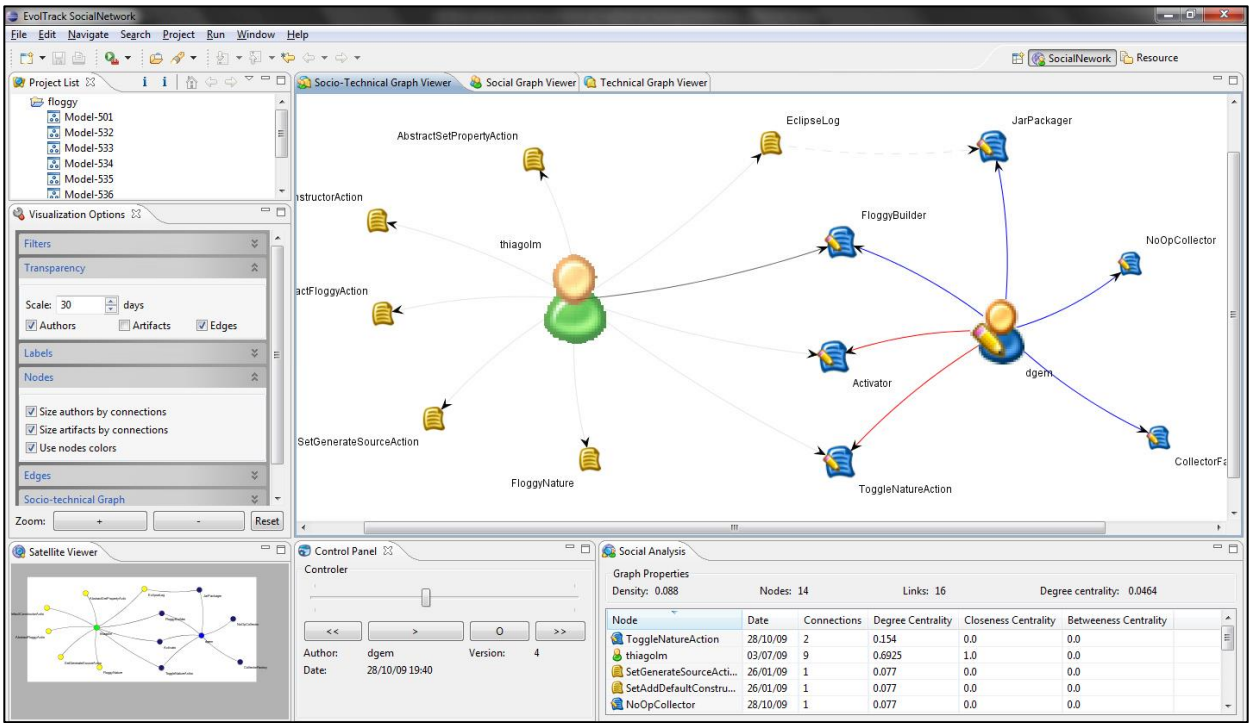

Figura 7 – Visão geral da interface da `EvolTrack-SocialNetwork` usando o projeto `Floggy`. Fonte: Adaptado de Magdaleno (*et al.*, 2010).

A `EvolTrack-SocialNetwork`, ilustrada na Figura 7, tem por objetivo oferecer percepção sobre como a colaboração acontece entre os membros de uma equipe de desenvolvimento de software. A percepção é o entendimento das atividades dos outros desenvolvedores para fornecer contexto a sua própria atividade (DOURISH; BELLOTTI, 1992). Desta forma, a ferramenta trabalha com

[20] **http://www.eclipse.org**

redes técnicas (apresenta a dependência técnica entre os artefatos), sócio-técnicas (relaciona os artefatos da rede técnica com os seus respectivos autores), sociais (apresenta a dependência social entre os autores dos artefatos) e disponibiliza recursos para melhorar a visualização destas redes e observar o seu comportamento dinâmico ao longo do tempo.

2.4.3 Tesseract

O Tesseract (SARMA *et al.*, 2009) é uma ferramenta que permite a exploração sócio-técnica a partir da análise dos arquivos de código, registros de comunicação e banco de dados do projeto para capturar as relações entre código, desenvolvedores e defeitos. No entanto, a ferramenta não explora plenamente a aplicação de métricas que tem um grande impacto na análise sócio-técnica.

A Figura 8 apresenta os principais componentes da Tesseract:

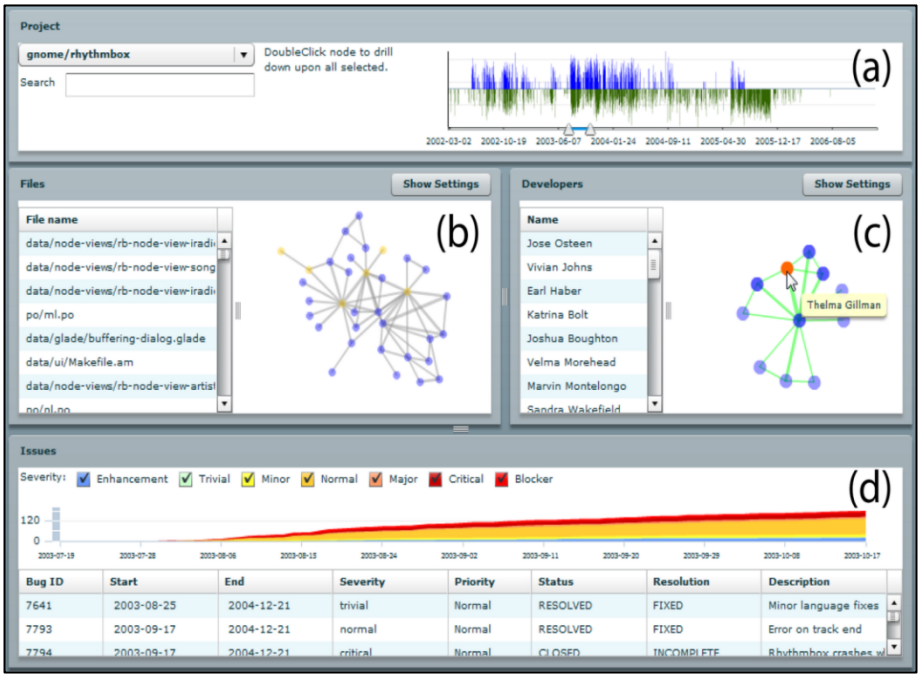

Figura 8 – Tela principal do Tesseract.

Fonte: Adaptado de Sarma (*et al.*, 2009).

a) Painel de Atividades de projeto: apresenta as atividades gerais de um projeto em uma linha do tempo. Ele permite ao usuário selecionar um período para análise que será refletido em todos os outros painéis.

b) Painel de rede de Arquivos: exibe as dependências dos artefatos como uma rede de arquivo para arquivo, baseando-se em arquivos que são frequentemente alterados em conjunto.

c) Painel de rede de Desenvolvedores: exibe as relações entre desenvolvedores. Dois desenvolvedores são considerados relacionados quando há registro de que fizeram alterações em um mesmo artefato ou em artefatos interdependentes.

d) Painel de *Issues*: exibe informações das *issues* sendo possível separar por tipos em um gráfico de área empilhada, bem como uma lista de detalhes.

A ferramenta `Tesseract` foi especificamente construída para:

- mostrar simultaneamente as relações sociais, bem como técnica entre entidades de projeto diferentes (por exemplo, os desenvolvedores, a comunicação, os códigos e os defeitos);

- ressaltar a ligação (ou a falta dela) entre as relações sociais e técnicas.

- vincular ligações e permitir a exploração interativa dessas relações e como elas mudam ao longo do tempo.

2.4.4 `BugMaps`

O `BugMaps` (HORA *et al.,* 2012) é uma ferramenta para a exploração visual e análise de defeitos que fornece mecanismos para automatizar o processo de análise de dados de repositórios de software. Ela possui algoritmos para mapear defeitos reportados nas classes de software em projetos orientados a objetos e fornece visualizações para auxiliar os desenvolvedores na tomada de decisão.

Esta ferramenta pode nos orientar a responder questões como: (a) Quais são os módulos envolvidos na correção de defeitos? (b) Qual é o ciclo de vida de um defeito? (c) Qual é o período que um módulo apresentou mais defeitos? (d) Quais módulos estão estáveis ou instáveis com relação à defeitos? (e) Quais são os módulos, cujo número de defeitos aumentou ou diminuiu ao longo do tempo? e (f) Qual é o número total de defeitos de um módulo?

O `BugMaps` recebe como entrada de dados arquivos de *log* de plataformas de controle de versão como `CVS` ou `SVN`, relatórios de defeitos de

plataformas de *bug-tracking (*ferramenta utilizadas para manter o controle de defeitos reportados durante o desenvolvimento de um projeto de software*)* como `JIRA` ou `Bugzilla` e realiza o mapeamento dos defeitos das classes atribuindo uma série temporal a ela.

Por fim, como mostrado na Figura 9, o `BugMaps` fornece visualizações interativas calculando métricas como: total de defeitos adicionados ou removidos dentre duas versões, evolução dos defeitos ao longo do tempo, total de defeitos adicionados ou removidos em versões subsequentes e também o número de versões existentes entre dois extremos que contenha pelo menos um defeitos.

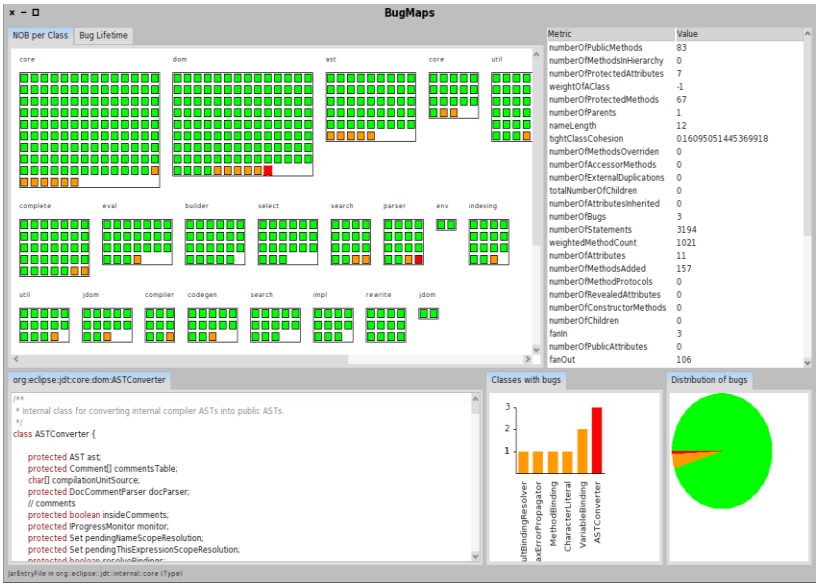

Figura 9 – Navegador instantâneo do `BugMaps`.

Fonte: Adaptado de COUTO (*et al.*, 2012).

2.5 TABELA DE COMPARAÇÃO ENTRE TRABALHOS RELACIONADOS

O Quadro 3 mostra a comparação entre as funcionalidades das ferramentas citadas no tópico de trabalhos relacionados, comparando com as funções da ferramenta proposta neste trabalho.

	Ferramentas relacionadas				
Funcionalidades principais da ferramenta proposta neste trabalho	xFlow	Tesseract	Evoltrack-SN	BugMaps	Ferramenta proposta
Mineração de dados técnicos	X	X	X	X	X
Mineração de dados sociais	X		X		X
Geração de redes técnicas	X	X	X		X
Geração de redes sociais		X	X		X
Geração de redes sócio-técnicas		X	X		X
Cálculo de métricas sociais					X
Cálculo de métricas técnicas	X			X	X
Exportação das redes sociais em arquivos CSV ou equivalentes.					X
Exportação das métricas em arquivos CSV ou equivalentes.				X	X
Adição de novas redes e métricas	X		X		X
Plataforma *web*		X			X

Quadro 3 – Comparação entre os trabalhos relacionados.

Fonte: Autoria própria.

2.6 TECNOLOGIAS UTILIZADAS

Nesta seção, serão abordadas as tecnologias escolhidas no desenvolvimento da ferramenta proposta, visando obter agilidade em sua construção explorando o máximo das tecnologias disponíveis, mas sem deixar de lado a qualidade dos resultados.

A linguagem de programação escolhida foi o JAVA, devido a gama de componentes disponíveis, ao conhecimento já adquirido sobre a sintaxe e também pelo fato do GitHub oferecer uma biblioteca escrita em JAVA de comunicação via JSON para coleta dos dados do site.

Para persistência dos dados optou-se pela biblioteca Eclipse-Link

JPA com o objetivo de tornar flexível a escolha do gerenciador de banco de dados. A interface foi desenvolvida em JSF[21] com componentes da biblioteca Prime Faces[22]. Para a implantação foi escolhido o servidor de aplicação GlassFish[23]. O ambiente integrado de desenvolvimento utilizado foi o NetBeans.

Optou-se pelo desenvolvimento de uma aplicação *web* pensando na possibilidade de disponibilizá-la para pesquisadores em locais geográficos distintos, de modo que possam compartilhar os dados coletados e a geração das redes e métricas.

[21] **http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html**
[22] **http://www.primefaces.org**
[23] **http://glassfish.java.net**

3 A FERRAMENTA

O objetivo deste trabalho é construir uma ferramenta que ofereça aos pesquisadores a possibilidade de efetuar a coleta de dados, geração de redes e cálculo de métricas para os repositórios de software hospedados no `GitHub`. A ferramenta em questão diferencia-se das demais, principalmente, pelas razões a seguir:

- poucos trabalhos oferecem uma ferramenta visual para coleta e análise de dados de repositórios de software hospedados no site do `GitHub`.

- a arquitetura foi desenvolvida para possibilitar a adição de novos algoritmos para geração de redes técnicas, sociais e cálculos de métricas.

Para possibilitar a implementação de novas métricas e redes para a ferramenta, três módulos foram projetados: o módulo de coleta, que utiliza uma API de comunicação via `JSON` disponibilizada pelo próprio `GitHub` para obtenção dos dados do site; o módulo de geração e exportação das redes; e o módulo para cálculo e exportação de métricas.

Os detalhes da arquitetura, características sobre a adição de novas métricas e redes, bem como exemplos de utilização dos módulos existentes são apresentados nas seções a seguir.

3.1 A ARQUITETURA DA FERRAMENTA

A arquitetura foi projetada com foco na possibilidade de adição de novas métricas e redes com o mínimo de esforço necessário. A Figura 10 ilustra esta divisão e a ligação entre os módulos. Tais módulos serão introduzidos nesta Seção e detalhados nas subseções seguintes.

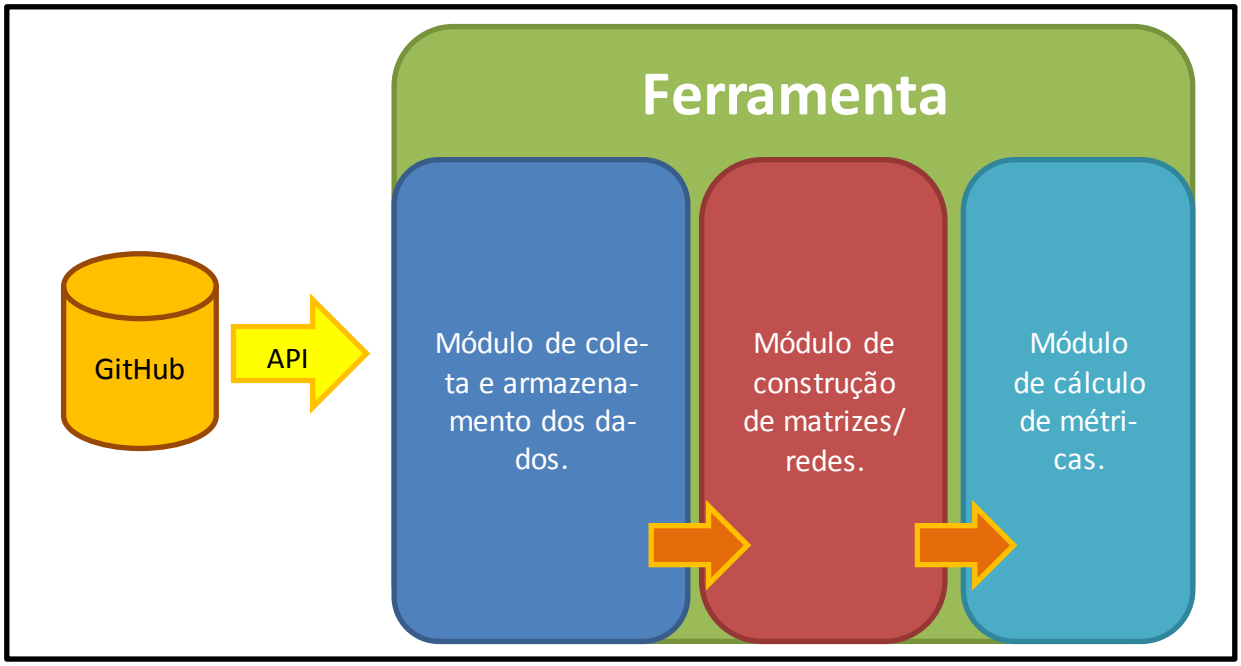

Figura 10 – Visão lógica da comunicação entre os módulos desenvolvidos neste trabalho.
Fonte: Autoria própria

O módulo de coleta é responsável por realizar a comunicação com o banco de dados do `GitHub` via `API` e coletar os dados do repositório escolhido pelo usuário. Os dados obtidos são persistidos em um banco de dados local através da biblioteca de persistência `JPA` e ficam disponíveis para utilização nos módulos seguintes.

Uma vez que os dados foram coletados, torna-se possível a utilização do módulo de geração das redes. Este possibilita que sejam criados algoritmos para geração das redes em forma de matrizes tendo como entrada os dados vindos do `GitHub`.

O terceiro e último módulo é responsável por calcular as métricas sobre as redes geradas. Através dele é possível implementar algoritmos para cálculo de métricas com base nos dados das redes. Os algoritmos podem utilizar como entrada resultados gerados pelo módulo de criação das redes.

Para prover o funcionamento dos módulos de geração das redes e cálculo de métricas de forma que possa ser incrementado facilmente, foi criada uma arquitetura genérica utilizando classes abstratas que padronizam as requisições, processamento dos dados e persistência dos resultados obtidos. Utilizando a orientação a objetos foi criado então um conceito de “classes *Services*”, em que uma

classe abstrata chamada de `AbstractServices` padroniza a forma de processamento das requisições de geração das matrizes e cálculo das métricas e é responsável por receber a requisição enviada pelo usuário, realizando o processamento e a persistência dos resultados em no formato `CSV` (*Comma-Separated Values*). Dependendo do módulo no qual são gerados, estes resultados podem representar uma rede ou uma métrica. Toda classe `Service` criada a partir de então deve, obrigatoriamente, estender a classe `AbstractServices`. Esta classe implementa a interface `java.lang.Runnable` e fica localizada dentro do pacote `br.edu.utfpr.cm.JGitMinerWeb.services`. Na Figura 11 é apresentada a classe mencionada utilizando a notação UML.

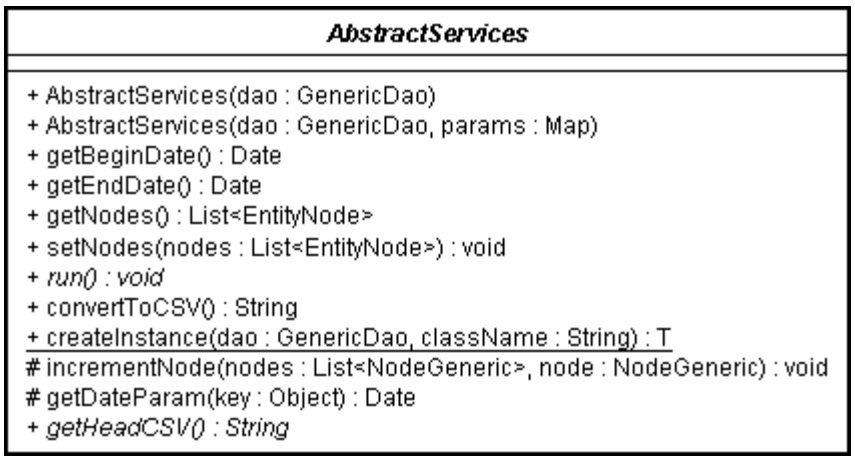

Figura 11 – Representação em UML da classe `Service` abstrata principal

Fonte: Autoria própria

Como visto na Figura 11, a classe `AbstractServices` possui um construtor que recebe como parâmetros o objeto de conexão com o banco de dados e um objeto contendo os filtros informados pelo usuário na página `JSF`. Estes objetos são:

– Um objeto do tipo `GenericDao` que possibilita a conexão com o banco de dados para leitura e persistência das informações;

– Um objeto do tipo `Map` contendo os filtros informados pelo usuário na página `JSF`.

Quando estendida, a classe `Service` principal força a criação de dois métodos abstratos que serão utilizados nas requisições, tanto de geração das redes

quanto de geração das matrizes, são eles:

– `run()`: é o método que recebe a requisição do usuário e deve conter o código fonte responsável por buscar as informações do banco de dados local, fazer as análises e transformações necessárias e persisti-las novamente no formato desejado;

– `getHeadCSV()`: responsável por retornar uma `String` com o cabeçalho do arquivo `CSV` representado por essa matriz. Ele é utilizado quando o usuário solicita o download do arquivo `CSV` do resultado de uma rede gerada.

Esta classe também implementa métodos auxiliares que são utilizados comumente entre os módulos. O método `getDateParam(Object)` converte uma `String` vinda da página `JSF` em uma objeto do tipo `Date`. O método `incrementNode(List<NodeGeneric>,NodeGeneric)` é responsável por adicionar um nó a lista de nós ou incrementar seu peso (*weigth*) caso ele já esteja adicionado na lista.

3.1.1 MÓDULO DE COLETA E ARMAZENAMENTO DOS DADOS

O módulo de coleta é responsável por iniciar o ciclo de análise dos dados na utilização da ferramenta. É por meio dele que se torna possível a obtenção dos dados dos repositórios de software hospedados no `GitHub` e persistência no banco de dados local. Este é o único módulo que não faz uso da classe `AbstractServices`, pois ele não utiliza o conceito de `Services` genéricos em suas requisições.

A coleta dos dados é feita por meio de uma biblioteca escrita em `JAVA` que se comunica via `JSON` com o banco de dados remoto do `GitHub`. Na biblioteca estão mapeadas as entidades do `GitHub` em forma de classe. É com base neste mapeamento que foram criadas as classes entidade do `JPA`. A Figura 12 apresenta a relação das classes mapeadas na ferramenta proposta com base na API fornecida pelo `GitHub`.

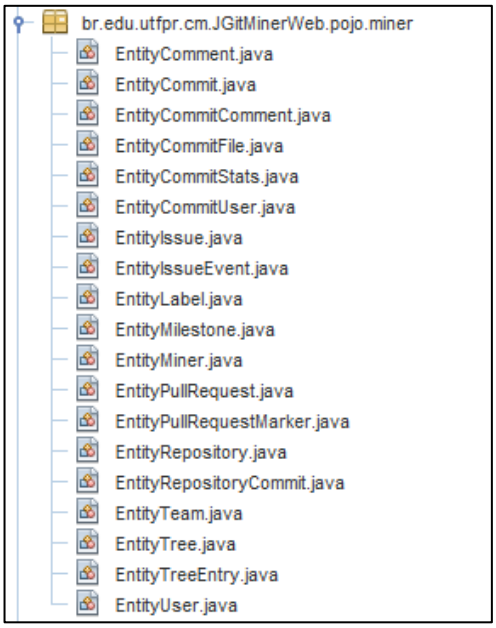

Figura 12 – Relação das classes entidade mapeadas com base na API do `GitHub`.

Fonte: Autoria própria

Como mostrado a seguir, dentre as classes entidades mapeadas, existem algumas com mais importância que se tornam um diferencial entre as informações existentes no repositório do `GitHub` para com os demais. Na Figura 13 é apresentado o diagrama contendo as principais classes as quais permitem a recuperação de aspectos sociais sobre o repositório de software.

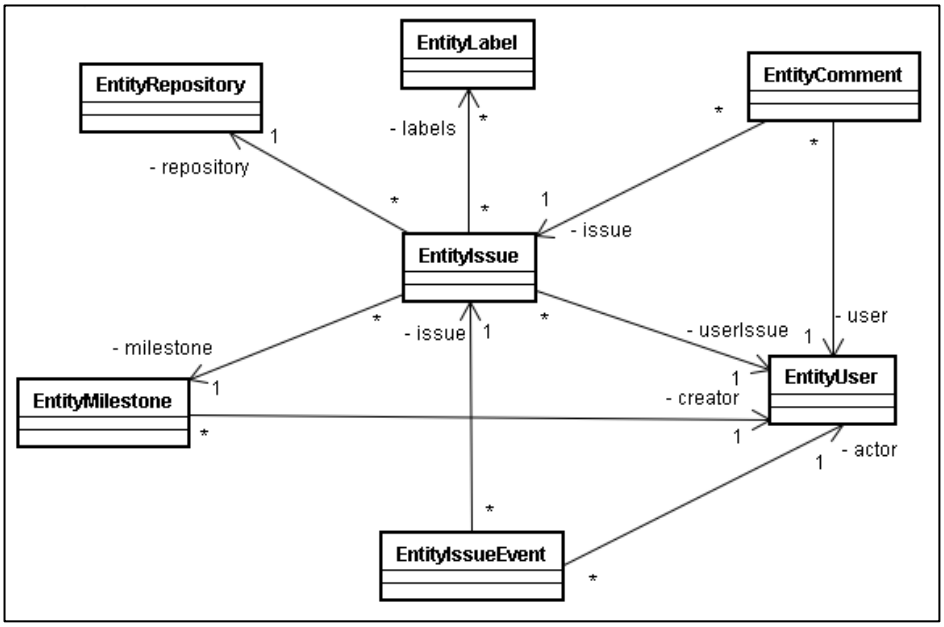

Figura 13 – Diagrama resumido das principais classes de aspecto social.

Fonte: Autoria própria

As classes da Figura 13 representam os seguintes conceitos no `GitHub`:

– `EntityRepository` – representa o repositório hospedado no `GitHub` que está sendo analisado.

– `EntityUser` – representa a conta de um usuário cadastrado no `GitHub`.

– `EntityMilestone` – representa um marco no desenvolvimento do projeto. Pode ser o lançamento de uma nova versão, uma mudança importante na arquitetura, entre outros.

– `EntityIssue` – representa o registro de uma tarefa a ser executada no projeto. Pode ser a correção de um erro, implementação de uma nova funcionalidade, entre outros.

– `EntityComment` – representa o comentário de um usuário naquela *issue*.

– `EntityLabel` – representa um rótulo ou anotação personalizada para aquela *issue*.

– `EntityIssueEvent` – representa um evento ocorrido naquela *issue*. Pode ser a criação da *issue*, fechamento, reabertura, cancelamento, entre outros.

A partir das ligações ilustradas na Figura 13, matrizes que representem redes de aspectos sociais do projeto podem ser geradas. Exemplos dessas matrizes são: os usuários que mais comentaram em uma determinada *issue*, usuários com maior quantidade de comentários em *issue* anotadas como “defeito”, usuários que realizam mais comentários com outros usuários, entre outros cenários. A partir destas redes, é possível chegar a conclusões, como por exemplo, poder afirmar que o projeto tende a apresentar defeitos quando um determinado usuário não faz comentários em uma *issue* anotada como “nova funcionalidade”.

Na Figura 14, é apresentado o diagrama resumido de classes que podem prover aspectos técnicos sobre o repositório de software analisado.

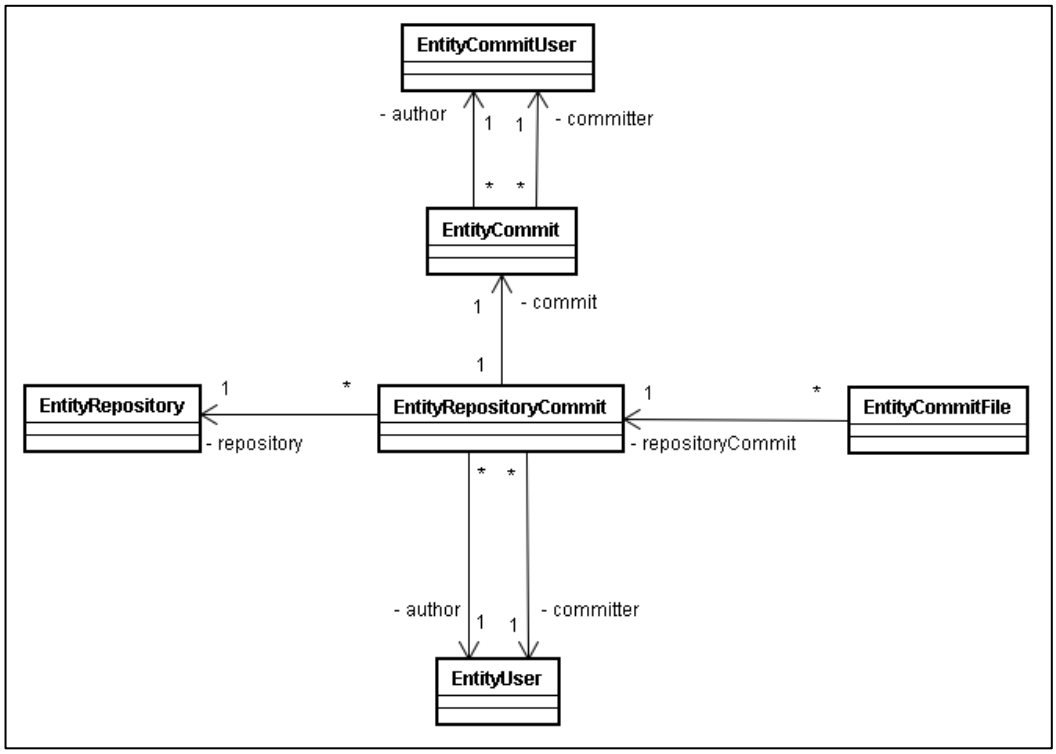

Figura 14 – Diagrama resumido das principais classes de aspecto técnico
Fonte: Autoria própria

As classes da Figura 14 representam os seguintes conceitos no `GitHub`:

– `EntityRepositoryCommit` – representa o evento do *commit* para o site do `GitHub`.

– `EntityCommit` – representa o evento do *commit* para o controlador de versão `Git`.

– `EntityCommitFile` – representa o arquivo adicionado, modificado ou removido por aquele `commit`.

– `EntityCommitUser` – representa o criador do *commit*, pode ser um usuário do `GitHub` ou um software com a posse da chave pública de autorização de acesso ao repositório em questão.

Estas classes possibilitam análises de aspectos técnicos sobre o repositório como, por exemplo, recuperar quais usuários modificaram mais vezes um determinado arquivo, quais usuários adicionaram ou removeram mais linhas dos arquivos, entre outros. Com isso, também é possível chegar a outros resultados, como, por exemplo, afirmar que quando um usuário modifica um arquivo ele tende a não receber novas modificações por um grande período futuro.

Também é possível recuperar informações para análises de aspectos sócio-técnicos. Estas relações são destacadas na Figura 15:

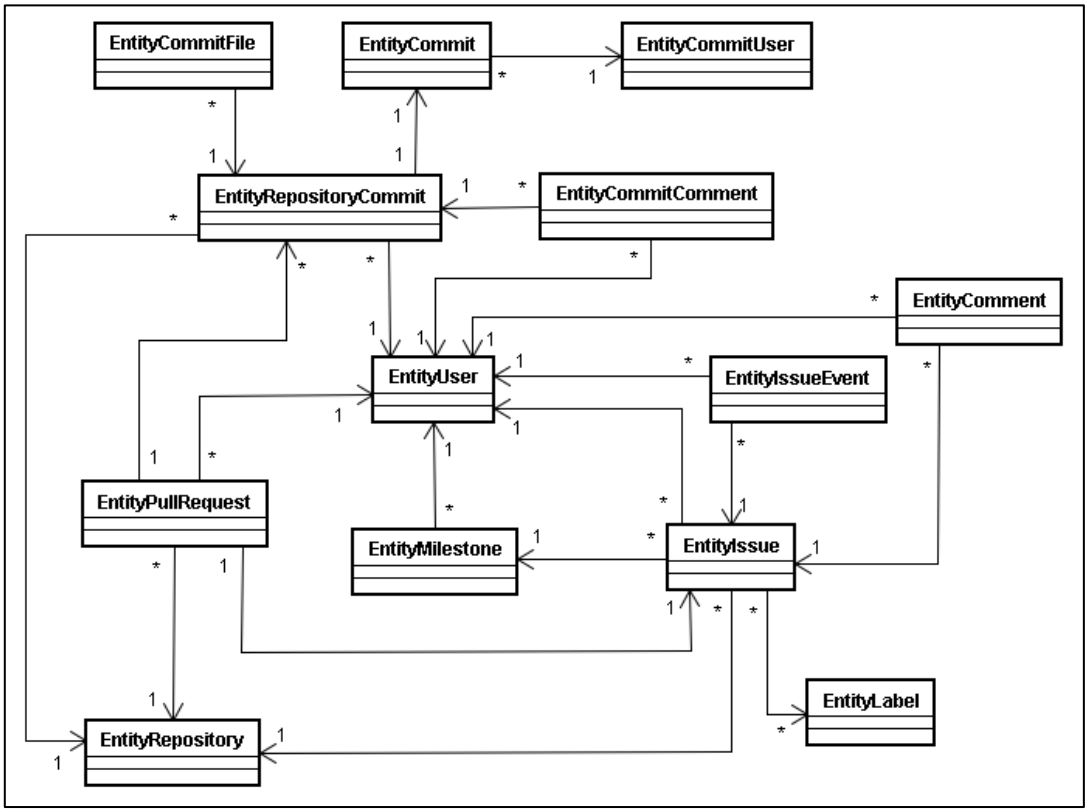

Figura 15 – Diagrama resumido de aspecto sócio-técnico contendo as principais classes (os nomes das relações foram omitidos para facilitar a visualização das ligações)

Fonte: Autoria própria

As classes da Figura 15 representam os seguintes conceitos no `GitHub`:

– `EntityPullRequest` – representa a submissão de um ou mais *patches* e pode conter um ou mais *commits*, cada *pull request* tem ligação única e exclusiva com uma *issue*.

– `EntityCommitComment` – representa o comentário de um usuário do `GitHub` em um *commit*, este comentário pode referenciar a linha de um arquivo ou o *commit* todo.

Como visto na seção 2.1, esta combinação entre aspectos técnicos e sociais proporcionam análises diferenciadas sobre o comportamento de um projeto de software, são as análises de aspectos sócio-técnicos.

O módulo de coleta dos dados oferece também opções para o usuário escolher quais as informações que ele deseja coletar do repositório de software do

GitHub. Estas opções de escolha são muito importantes, pois dependendo do tamanho do projeto, o processo de coleta dos dados pode levar várias horas ou até dias para ser concluído. Desta forma, o usuário pode escolher somente os dados que são importantes para a sua análise e, assim, diminuir o tempo de espera. Por exemplo, se o usuário tem como objetivo gerar uma rede social com bases nos comentários dos desenvolvedores em uma *issue*, ele poderá então coletar somente os dados das *issues* e seus comentários, deixando de lado os dados dos *commits*, colaboradores, *forks*, etc. A Figura 16 mostra a página de coleta dos dados do repositório.

Figura 16 – Página de coleta dos dados dos repositórios de software hospedados no GitHub

Fonte: Autoria própria

3.1.2 MÓDULO DE GERAÇÃO DAS REDES

O módulo de geração das redes foi criado para permitir a adição de novos algoritmos de forma simplificada. A arquitetura baseada em "classes `Services`" possibilita tal ação sem necessitar de alteração de código fonte já existente na estrutura da ferramenta.

Criou-se então, exclusivamente para o módulo de geração das redes, uma classe abstrata chamada `AbstractMatrizServices` que herda a classe principal `AbstractServices` e fica localizada dentro do pacote `br.edu.utfpr.cm.JGitMinerWeb.services.matriz`. Ela é responsável por padronizar o processo de geração das redes e também implementar um método específico chamado `addToEntityMatrizNodeList(List)`, que recebe como parâmetro uma lista de resultados parciais e adiciona este a lista de resultados finais que darão origem a rede. A Figura 17 ilustra esta classe.

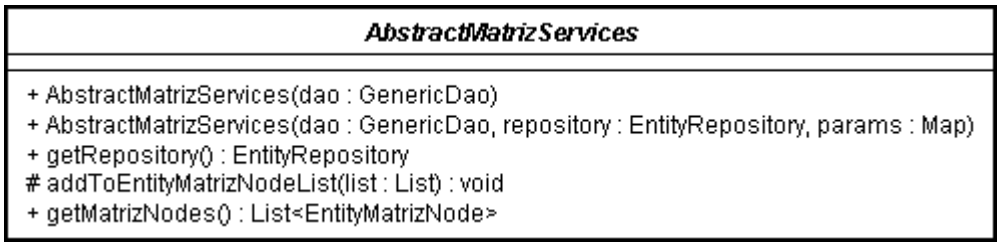

Figura 17 – Representação em UML da classe `Service` abstrata de geração das redes
Fonte: Autoria própria

Em relação à classe `Service` principal, há uma diferença em um de seus construtores, foi adicionado o parâmetro `EntityRepository`, que é responsável por trazer o repositório selecionado pelo usuário na página de geração das redes.

Com essa arquitetura, fica simples a implementação de uma nova classe para geração de redes, basta criar uma classe `Service` estendendo a abstrata `AbstractMatrizServices` dentro do pacote `br.edu.utfpr.cm.JGitMinerWeb.services.matriz`. Feito isso, quando o usuário acessar a página de geração das redes, aparecerá automaticamente a nova classe como opção de escolha. Esta automatização é feita utilizando a API `Java`

`Reflection`[24].

Nesta nova classe criada, é necessário implementar dois métodos: no método `run()` o usuário deve escrever o algoritmo de geração da rede da forma que desejar. Esse algoritmo terá que ler os dados do banco de dados local, processar as informações de sua preferencia e, após obter o resultado esperado, passa-lo para o método `addToEntityMatrizNodeList(List)` em forma de um objeto `List` para que sejam persistidos novamente no banco de dados; e no método `getHeadCSV():String` é preciso escrever, em forma de `String`, como será o cabeçalho do arquivo `CSV` desta rede.

Por fim, se necessário, é possível criar uma página `JSF` dentro do diretório `web/pages/matriz/filter` que conterá os campos para os filtros desejados na geração da rede, tal como informar o número de um *pull request*, intervalo de data, etc. Essa página deve ter exatamente o mesmo nome da classe `Service` criada pelo usuário anteriormente e então os filtros aparecerão automaticamente na tela, através de um `include`, quando o usuário selecionar a respectiva classe `Service` para geração da rede. Os filtros são passados da página `JSF` para a classe `Service` através de um objeto do tipo `Map`. A Figura 18 mostra a página de geração das redes e os filtros exibidos quando uma classe `Service` é selecionada.

[24] **http://docs.oracle.com/javase/tutorial/reflect/**

Figura 18 – Página de geração das redes

Fonte: Autoria própria

Conforme ilustrado na Figura 18, para dar início a geração de uma rede, o usuário precisa selecionar primeiro o repositório desejado, em seguida a classe `Service` que irá gerar a rede desejada, e por último, informar os filtros conforme sua preferência.

3.1.3 MÓDULO DE CÁLCULO DAS MÉTRICAS

Exatamente como o módulo de geração de redes, o módulo de cálculo de

métricas foi projetado de forma que possa ser estendido facilmente. Assim foi criada uma classe abstrata chamada `AbstractMatrizServices` que herda a classe principal `AbstractServices` e fica localizada dentro do pacote `br.edu.utfpr.cm.JGitMinerWeb.services.metric`. Ela é responsável por padronizar o processo de cálculo das métricas e também implementar um método específico chamado `addToEntityMetricNodeList(List)`, que recebe como parâmetro uma lista de resultados parciais e adiciona-os à lista de resultados finais. A Figura 19 ilustra esta classe.

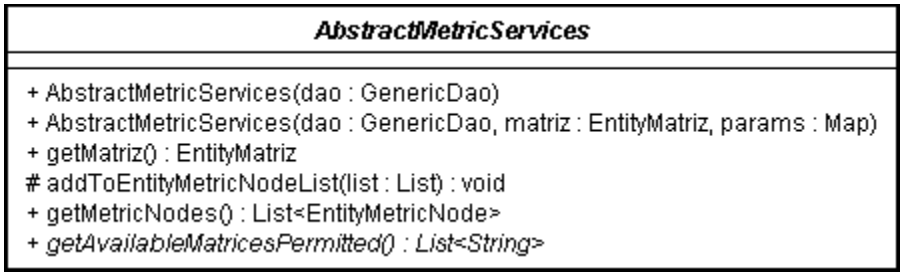

Figura 19 – UML da classe `Service` abstrata de cálculo das métricas

Fonte: Autoria própria

Assim como no módulo de geração das redes, a classe `AbstractMetricServices` possui um parâmetro adicional do tipo `EntityMatriz` em um de seus construtores, que é responsável por trazer o resultado de uma rede gerada anteriormente e que servirá como entrada de dados para o cálculo das métricas. Acrescenta também mais um método abstrato `getAvailableMatricesPermitted():List<String>`.

Para adicionar uma nova classe para geração de redes, basta criar uma classe `Service` estendendo a abstrata `AbstractMetricServices` dentro do pacote `br.edu.utfpr.cm.JGitMinerWeb.services.metric`. Feito isso, quando o usuário acessar a página de cálculo das métricas, aparecerá automaticamente a nova classe como opção de escolha.

Neste módulo, ao adicionar uma classe Service o usuário precisará implementar três métodos. No método `run()` o usuário deve escrever o algoritmo que calcula as métricas que desejar. Este algoritmo receberá como entrada o resultado de uma rede em forma de matriz, processará as informações conforme sua preferência e, após obter o resultado esperado, passá-lo para o método `addToEntityMetricNodeList(List)` em forma de um objeto `List`. No método

`getHeadCSV():String` é preciso escrever em forma de `String` como será o cabeçalho do arquivo `CSV` que conterá o resultado desta métrica. No método `getAvailableMatricesPermitted():List<String>` deverá retornar uma lista com os nomes das classes `Service` de geração de redes que poderão servir como entrada de dados para esta métrica.

Por fim, se necessário, é possível criar uma página `JSF` dentro do diretório `web/pages/metric/filter` que conterá os campos para os filtros desejados no cálculo da métrica. Essa página deve ter exatamente o mesmo nome da classe `Service` respectiva criada anteriormente e então os filtros aparecerão automaticamente na tela, através de um `include`, quando o usuário selecionar a respectiva classe `Service`. Os filtros são passados da página `JSF` para a classe `Service` através de um objeto do tipo `Map`. A Figura 20 mostra a página de cálculo das métricas e os filtros exibidos quando uma classe `Service` é selecionada.

Figura 20 – Página de cálculo de métricas

Fonte: Autoria própria

Conforme ilustrado na Figura 20, para dar início ao processo de cálculo de uma métrica, o usuário precisa selecionar primeiro a rede que servirá como entrada de dados. Então a lista de classes `Service` será atualizada com as métricas disponíveis para esta rede selecionada, em seguida selecionar a classe `Service` que irá calcular a métrica desejada e, por último, informar os filtros conforme sua preferência.

3.2 FUNCIONAMENTO DA ARQUITETURA, EXEMPLO DE IMPLEMENTAÇÃO E ADIÇÃO DE NOVAS REDES E MÉTRICAS

Nesta seção serão apresentados, o funcionamento da arquitetura da ferramenta, o fluxo de execução, os relacionamentos entre os módulos e os detalhes de como devem ser adicionados as métricas e redes a esta ferramenta.

3.2.1 FUNCIONAMENTO DA ARQUITETURA

Todo processo se inicia com o armazenamento dos dados do repositório desejado a partir do módulo de coleta. Nesta coleta, o primeiro passo é cadastrar o repositório de software desejado informando o nome de usuário do dono do projeto e também o nome do projeto hospedado no `GitHub`. Tomamos como exemplo o projeto `Bootstrap`[25]: o nome de usuário do dono do repositório é "**twitter**" e o nome do repositório é "**bootstrap**".

Uma vez que o repositório foi cadastrado na ferramenta, ele passa a estar disponível para ser utilizado pelo módulo de geração das redes. Em seguida é preciso acessar a página de coleta dos dados ilustrada na Figura 16 da seção 3.1.1, escolher o repositório desejado, escolher os dados que deverão ser coletados e clicar no botão "*Start*". A Figura 21 mostra a página de cadastro de repositórios na

[25] **https://github.com/twitter/bootstrap**

ferramenta.

Figura 21 – Página de cadastro de repositórios

Fonte: Autoria própria

Quando a coleta é disparada, a ferramenta envia a requisição da página `JSF` a um `ManagedBean` de sessão via `AJAX` (*Asynchronous Javascript and XML*), que por sua vez irá executar todo o processo a partir de uma `Thread`. Dessa forma a ferramenta não fica presa até o término da coleta e possibilita ao usuário navegar em outras páginas sem interromper o serviço. A partir deste ponto começa-se a coleta dos dados buscando as informações no banco de dados do `GitHub` e armazenando no banco de dados local. O processo estará concluído quando a barra de progresso chegar a **100%**.

Tendo efetuado a coleta dos dados, a próxima etapa é a utilização do módulo de geração das redes. Para isso deve-se acessar a página de geração das redes conforme mostra a Figura 18 da seção 3.1.2, escolher o repositório desejado, escolher a classe `Service` que irá gerar a rede desejada, informar os filtros de sua preferência caso necessário e clicar no botão "*Start*".

Da mesma forma que no módulo de coleta, será enviada uma requisição via `AJAX` da página `JSF` para um `ManagedBean` de sessão, que por sua vez irá

executar todo o processo dentro de uma `Thread`.

A partir de então a ferramenta inicia o processo de geração da rede: primeiro é criada uma instância da classe `Service` escolhida pelo usuário, a partir de seu nome, utilizando `Java Reflection`; em seguida são passados para essa instância o objeto `GenericDao` que fornece conexão com o banco de dados local, o objeto `EntityRepository` que representa o repositório selecionado pelo usuário e um objeto `Map` contendo os filtros informados na página `JSF`; na sequência é chamado o método `run()` da classe `Service`, para que o algoritmo de geração da rede seja executado; e por fim, o resultado é persistido localmente no banco de dados em formato de arquivo `CSV`. Nesta etapa o usuário já terá acesso ao *download* das redes geradas em formato de arquivo `CSV` como mostra a `Figura 22`.

Figura 22 – Página de listagem das redes geradas

Fonte: Autoria própria

A terceira e última etapa deste clico de funcionamento da ferramenta se dá através do módulo de cálculo das métricas.

Para realizar o cálculo de uma métrica implementada, primeiramente deve-se acessar a página de cálculo das métricas mostrada na Figura 20 na sessão 3.1.3; em seguida é preciso escolher uma rede já gerada anteriormente que servirá

como entrada de dados. Feito isso a lista de classes `Service` de cálculo de métricas será atualizada, exibindo somente as classes `Service` disponíveis para aquela rede selecionada; na sequência selecionar a classe Service desejada; depois preencher os filtros caso necessário; e por último, clicar no botão "*Start*".

As etapas arquiteturais deste último módulo são basicamente as mesmas do módulo de geração das redes. A página `JSF` envia a requisição via `AJAX` para um `ManagedBean` de sessão que executa o processo dentro de uma `Thread`. Neste processo: é criada uma instância da classe `Service` via `Java Reflection`; é passada para esta instância um objeto `GenericDao`, `Map`, e um objeto `EntityMatriz` contendo os dados da rede escolhida pelo usuário para servir como entrada de dados; chama-se o método `run()` para execução do algoritmo que calculará as métricas; e por fim, o resultado é armazenado localmente no banco de dados em formato de arquivo `CSV`.

Figura 23 – Página listagem das métricas calculadas

Fonte: Autoria própria

Como no módulo anterior, neste ponto o usuário já terá acesso ao

download dos resultados das métricas calculadas em formato de arquivo `CSV` como mostra a Figura 23.

3.2.2 EXEMPLO DE IMPLEMENTAÇÃO DE REDES E MÉTRICAS

Conforme descrito nas seções 3.1.2 e 3.1.3, foi criado na ferramenta um conceito de classes `Services`, utilizando abstração, para prover facilidade ao estender os módulos de geração das redes e cálculo das métricas. Nesta seção serão apresentados exemplos de implementação de uma classe `Service` para geração de rede e outra para cálculo de métrica.

A fim de validar a adição de métricas e redes para a ferramenta, foi replicado o artigo ***Predicting Failures with Developer Networks and Social Network Analysis*** (MENEELY *et al*., 2008). Neste estudo foi gerada uma rede sócio-técnica em que os nós (*vertices*) são os desenvolvedores e as arestas (*edges*) são os arquivos modificados dentro de uma *release*. Em seguida calculadas métricas de grau (*degree*), proximidade (*closeness*) e intermediação (*betweenness*) para os desenvolvedores.

```
public class UserModifySameFileInMilestoneServices
        extends AbstractMatrizServices {

    public UserModifySameFileInMilestoneServices(GenericDao dao) {
        super(dao);
    }

    public UserModifySameFileInMilestoneServices(GenericDao dao,
            EntityRepository repository, Map params) {
        super(dao, repository, params);
    }

    @Override
    public void run() {
    }

    @Override
    public String getHeadCSV() {
        return "";
    }
}
```

Figura 24 – Código mínimo da classe `UserModifySameFileInMilestoneServices`

Fonte: Autoria própria

Primeiro é necessário implementar o `Service` de geração da rede. Veja

na Figura 24 o código mínimo para a classe `Service` criada. A chamamos de `UserModifySameFileInMilestoneServices`. Esta deve herdar a classe abstrata `AbstractServices` e ficar dentro do pacote `br.edu.utfpr.cm.JGitMinerWeb.services.matriz`. Assim esta classe será responsável por gerar a rede sócio-técnica, onde os nós serão os desenvolvedores e as arestas serão os arquivos modificados dentro de um *milestone*.

Desta forma, ao acessar a página de geração de redes já terá disponível esta nova classe `Service` para ser selecionada. O próximo passo é implementar no método `run()` o algoritmo que irá gerar a rede, conforme Figura 25.

```
@Override
public void run() {
    if (getRepository() == null) {
        throw new IllegalArgumentException("Parâmetro Repository não pode ser nulo.");
    }

    String jpql = "SELECT DISTINCT NEW "
            + AuxUserUserFile.class.getName() + "(rc.committer.login, rc2.committer.login, f.filename) "
            + "FROM "
            + "EntityPullRequest p JOIN p.repositoryCommits rc JOIN rc.files f, "
            + "EntityPullRequest p2 JOIN p2.repositoryCommits rc2 JOIN rc2.files f2 "
            + "WHERE "
            + "p.repository = :repository AND "
            + "p2.repository = :repository AND "
            + "p.issue.milestone.number = :milestoneNumber AND "
            + "p2.issue.milestone.number = :milestoneNumber AND "
            + "f.filename = f2.filename AND "
            + "rc.committer.login <> rc2.committer.login";

    String[] bdParams = new String[]{
        "repository",
        "milestoneNumber"
    };
    Object[] bdValues = new Object[]{
        getRepository(),
        getMilestoneNumber()
    };

    List<AuxUserUserFile> result = dao.selectWithParams(jpql, bdParams, bdValues);

    result = removeDuplicate(result);

    addToEntityMatrizNodeList(result);
}
```

Figura 25 – Implementação do método `run()` da classe `UserModifySameFileInMilestoneServices`

Fonte: Autoria própria

O método `run()` é composto de uma consulta no banco de dados local, que irá selecionar e unir os desenvolvedores que modificaram o mesmo arquivo

dentro do *milestone*; da utilização de uma classe auxiliar chamada `AuxUserUserFile` que armazenará o resultado parcial da consulta em uma lista; da chamada de um método auxiliar que remove registros duplicados da lista; e para concluir, a lista de resultados é passada ao método `addToEntityMatrizNodeList(List)` que os devolverá ao `ManagedBean` para serem salvos em formato `CSV`.

Para finalizar a implementação da classe `Service` de geração da rede, é preciso escrever o método `getHeadCSV()`, que deve retornar o cabeçalho das colunas do arquivo `CSV`. Para o exemplo em questão será retornado "`desenvolvedor;desenvolvedor2;arquivo`". Veja a Figura 26.

```
@Override
public String getHeadCSV() {
    return "desenvolvedor;desenvolvedor2;arquivo";
}
```

Figura 26 – Implementação do método `getHeadCSV()` da classe `UserModifySameFileInMilestoneServices`

Fonte: Autoria própria

Uma vez que se tem a classe `Service`, pode-se criar o arquivo `JSF` que conterá os filtros que o usuário poderá informar no momento de gerar a rede. No exemplo em questão será necessário criar um campo para que seja informado o número do *milestone* desejado. Veja a Figura 27.

```
<div xmlns="http://www.w3.org/1999/xhtml"
     xmlns:h="http://java.sun.com/jsf/html"
     xmlns:p="http://primefaces.org/ui">
    <h2>Usuarios que modificaram o mesmo arquivo no milestone</h2>
    <br />
    <h:panelGrid columns="2" cellpadding="5" columnClasses="col">
        <h:outputText value="Milestone Number: " />
        <p:inputText value="#{gitMatrizBean.paramValue['milestoneNumber']}"
                     type="number"  size="10" />
    </h:panelGrid>
</div>
```

Figura 27 – Implementação do arquivo `JSF` contendo os filtros para geração da rede

Fonte: Autoria própria

Este arquivo deve ficar dentro do diretório `/pages/matriz/filter/` e deve ter exatamente o mesmo nome da classe `Service` correspondente, chamando também `UserModifySameFileInMilestoneServices`.

Assim conclui-se a adição de uma nova a rede, sem necessitar de alteração de código fonte em outras classes que não seja `UserModifySameFileInMilestoneServices`. A Figura 28 mostra parte do resultado de geração da rede em um arquivo `CSV` de 900 linhas gerado por esta nova classe `Service`.

	A	B	C
1	desenvolvedor	desenvolvedor2	arquivo
2	loalf	nfx	src/Symfony/Bridge/Twig/compos
3	nfx	loalf	src/Symfony/Component/Security
4	loalf	nfx	src/Symfony/Component/Depend
5	fabpot	loalf	src/Symfony/Component/Validatc
6	drak	loalf	src/Symfony/Component/Depend
7	nfx	loalf	src/Symfony/Component/DomCra
8	cordoval	loalf	src/Symfony/Component/Config/I
9	loalf	fabpot	src/Symfony/Bundle/WebProfilerI
10	loalf	nfx	src/Symfony/Bundle/TwigBundle/
11	kufi	loalf	src/Symfony/Bridge/Propel1/Test
12	nfx	jfsimon	src/Symfony/Component/Finder/T
13	nfx	jfsimon	src/Symfony/Component/Finder/I
14	fabpot	loalf	src/Symfony/Component/Validatc

Figura 28 – Parte do arquivo `CSV` da rede de desenvolvedores que modificaram o mesmo arquivo no *milestone* número 7 do projeto `Symfony`[26]
Fonte: Autoria própria

O próximo passo na reaplicação do estudo de (MENEELY *et al.*, 2008) é a implementação das métricas para desenvolvedores e para arquivos. Para calcular tais métricas foi utilizada a API da biblioteca `JUNG`[27].

Semelhante aos passos para geração da rede, cria-se a classe `Service` para geração das métricas, que deve herdar a classe `AbstractMetricServices`, deve ficar no pacote `br.edu.utfpr.cm.JGitMinerWeb.services.metric` e seu nome será `UserBetweenessDegreeClosenessServices`. A Figura 28 ilustra seu código mínimo.

[26] **https://github.com/symfony/symfony/**
[27] **http://jung.sourceforge.net/doc/**

```java
public class UserBetweenessDegreeClosenessServices
        extends AbstractMetricServices {

    public UserBetweenessDegreeClosenessServices(GenericDao dao) {
        super(dao);
    }

    public UserBetweenessDegreeClosenessServices(GenericDao dao,
            EntityMatriz matriz, Map params) {
        super(dao, matriz, params);
    }

    @Override
    public void run() {
    }

    @Override
    public String getHeadCSV() {
        return "";
    }

    @Override
    public List<String> getAvailableMatricesPermitted() {
        return null;
    }
}
```

Figura 29 – Código mínimo da classe `UserBetweenessDegreeClosenessServices`
Fonte: Autoria própria

Desta forma, ao acessar a página de cálculo das métricas já terá disponível esta nova classe `Service` para ser selecionada. Na sequência deve-se implementar no método `run()` o algoritmo que irá calcular as métricas, conforme Figura 30.

Este método `run()` é composto de um laço de repetição utilizado para preencher o objeto `Graph` da biblioteca `JUNG` com os dados provenientes da rede selecionada pelo usuário; de um laço de repetição que percorre os nós do objeto `Graph` calculando as métricas de grau (*degree*), proximidade (*closeness*) e intermediação (*betweenness*) para os desenvolvedores; de uma lista de objetos do tipo `AuxUserMetrics` que armazena o resultado parcial das métricas; de outro laço de repetição que calcula quais os desenvolvedores são *hubs*, ou seja, aqueles que tem o grau (*degree*) acima da média; e por fim a chamada do método `addToEntityMetricNodeList(List)` que devolverá os resultados ao `ManagedBean` para serem salvos em formato `CSV`.

```java
@Override
public void run() {
    if (getMatriz() == null
            || !getMatriz().getClassServicesName().
            equals(UserModifySameFileInMilestoneServices.class.getName())) {
        throw new IllegalArgumentException("Selecione uma matriz gerada pelo Service:
                + UserModifySameFileInMilestoneServices.class.getName());
    }

    UndirectedSparseGraph<String, String> graph = new UndirectedSparseGraph<>();
    for (int i = 0; i < getMatriz().getNodes().size(); i++) {
        EntityMatrizNode node = getMatriz().getNodes().get(i);
        String[] coluns = node.getLine().split(JsfUtil.TOKEN_SEPARATOR);
        if (coluns.length >= 3) {
            graph.addEdge(coluns[2] + " (" + i + ")", coluns[1], coluns[0]);
        }
    }

    BetweennessCentrality<String, String> btw = new BetweennessCentrality<>(graph);
    ClosenessCentrality<String, String> cls = new ClosenessCentrality<>(graph);
    DegreeScorer<String> dgr = new DegreeScorer<>(graph);

    List<AuxUserMetrics> userMetrics = new ArrayList<>();
    Double degreeAverage = 0d;
    int indexDegree = 2;
    int indexHub = 4;
    for (String vertex : graph.getVertices()) {
        AuxUserMetrics aux = new AuxUserMetrics(vertex,
                btw.getVertexScore(vertex),
                dgr.getVertexScore(vertex),
                cls.getVertexScore(vertex),
                0);
        userMetrics.add(aux);
        degreeAverage += aux.getMetrics()[indexDegree];
    }
    degreeAverage = degreeAverage / userMetrics.size();

    for (AuxUserMetrics aux : userMetrics) {
        if (aux.getMetrics()[indexDegree] > degreeAverage) {
            aux.getMetrics()[indexHub] = 1;
        }
    }

    addToEntityMetricNodeList(userMetrics);
}
```

Figura 30 – Implementação do método `run()` da classe `UserBetweenessDegreeClosenessServices`

Fonte: Autoria própria

Para finalizar a implementação da classe `Service` que calcula as

métricas para os desenvolvedores, é preciso escrever ainda os método `getHeadCSV()` que deve retornar o cabeçalho das colunas do arquivo `CSV` e o método `getAvailableMatricesPermitted()` que retorna a lista de classes `Service` de geração de redes para o qual a métrica terá validade. Veja a Figura 31.

```
@Override
public String getHeadCSV() {
    return "user;betweeness;degree;closeness;hub";
}

@Override
public List<String> getAvailableMatricesPermitted() {
    return Arrays.asList(UserModifySameFileInMilestoneServices.class.getName());
}
```

Figura 31 – Implementação dos métodos `getHeadCSV()` e `getAvailableMatricesPermitted()` da classe `UserBetweenessDegreeClosenessServices`

Fonte: Autoria própria

Assim conclui-se a implementação da classe `Service` para cálculo de métricas para desenvolvedores. Neste caso, não será necessária a criação do arquivo `JSF` para os filtros. A Figura 32 mostra parte do resultado das métricas calculadas em um arquivo `CSV` de 50 linhas gerado por esta nova classe `Service`.

	A	B	C	D	E
1	user	betweeness	degree	closeness	hub
2	loalf	815,4166667	48	1	1
3	vicb	0	2	0,510638298	0
4	nfx	223,9166667	33	0,761904762	1
5	leek	0	2	0,510638298	0
6	brikou	0	3	0,516129032	0
7	meandmymonkey	1	7	0,539325843	1
8	pulse00	0	2	0,510638298	0
9	kufi	0	1	0,505263158	0
10	beberlei	0	1	0,505263158	0
11	adrienbrault	0	1	0,505263158	0
12	romainneutron	2,583333333	7	0,539325843	1
13	dirkaholic	0	3	0,516129032	0
14	fabpot	9,416666667	11	0,564705882	1

Figura 32 – Parte do resultado das métricas para desenvolvedores calculadas pela classe `UserBetweenessDegreeClosenessServices`

Fonte: Autoria própria

A terceira e última etapa de replicação do estudo, citado no início desta seção, é a implementação da segunda classe `Service` para geração de métricas, que também receberá como entrada a rede gerada pela classe `UserModifySameFileInMilestoneServices`, porém desta vez serão calculadas métricas para os arquivos.

Via de regra, cria-se um `Service` que herda a classe `AbstractMetricServices` dentro do pacote `br.edu.utfpr.cm.JGitMinerWeb.services.metric` que chamamos de `FileBetweenessDegreeClosenessServices`. Esta classe deverá calcular as seguintes métricas para os arquivos:

- Código alterado (*Code churn*): O número de linhas de código que foram ou adicionados ou alterados ao longo da história deste arquivo.
- Atualizações (*Updates*): O número de atualizações para o repositório que incluiu este arquivo.
- Desenvolvedores (*Developers*): O número de desenvolvedores distintos que atualizaram este arquivo longo de sua história.
- Soma/Média/Máximo do grau (*Degree*): A soma, média e máximo do grau de cada desenvolvedor sobre a história de um arquivo.
- Soma/Média/Máximo de proximidade (*Closeness*): A soma, média e máximo da proximidade de cada desenvolvedor sobre a história de um arquivo.
- Soma/Média/Máximo de intermediação (*Betweenness*): A soma, média e máximo da intermediação de cada desenvolvedor sobre a história de um arquivo.
- Desenvolvedores Hub (*Hub Developers*): O número de desenvolvedores hub distintos que atualizam esse arquivo.

A implementação do método `run()` deverá conter os mesmos cálculos das métricas utilizado na classe `Service` para desenvolvedores acrescentando a complexidade dos cálculos para os arquivos. A Figura 33 mostra parte do resultado do arquivo `CSV` de 580 linhas das métricas geradas pela classe `FileBetweenessDegreeClosenessServices`.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	arquivos	btwMax	btwAve	btwSum	dgrMax	dgrAve	dgrSum	clsMax	clsAve	clsSum	developers	codeChurn	updates	bugs	hubs
2	src/Symfony/Bridge/Twi	815,4166667	349,5833333	1048,75	48	30,66666667	92	1	0,775536881	2,326610644	3	61	4	0	3
3	src/Symfony/Componen	815,4166667	519,6666667	1039,333333	48	40,5	81	1	0,880952381	1,761904762	2	56	3	0	2
4	src/Symfony/Componen	815,4166667	519,6666667	1039,333333	48	40,5	81	1	0,880952381	1,761904762	2	14	3	0	2
5	src/Symfony/Componen	815,4166667	274,9444444	824,8333333	48	20,66666667	62	1	0,693611638	2,080834915	3	249	3	0	2
6	src/Symfony/Componen	815,4166667	346,4444444	1039,333333	48	28,33333333	85	1	0,761214631	2,283643892	3	34	5	0	2
7	src/Symfony/Componen	815,4166667	346,4444444	1039,333333	48	27,66666667	83	1	0,757514353	2,27254306	3	30	3	0	2
8	src/Symfony/Componen	815,4166667	407,7083333	815,4166667	48	25	50	1	0,755319149	1,510638298	2	90	6	0	1
9	src/Symfony/Bundle/We	815,4166667	412,4166667	824,8333333	48	29,5	59	1	0,782352941	1,564705882	2	4	2	0	2
10	src/Symfony/Bundle/Tw	815,4166667	519,6666667	1039,333333	48	40,5	81	1	0,880952381	1,761904762	2	12	2	0	2
11	src/Symfony/Bridge/Pro	815,4166667	407,7083333	815,4166667	48	24,5	49	1	0,752631579	1,505263158	2	250	6	0	1
12	src/Symfony/Componen	815,4166667	346,4444444	1039,333333	48	27,66666667	83	1	0,757514353	2,27254306	3	112	3	0	2
13	src/Symfony/Componen	815,4166667	346,4444444	1039,333333	48	27,66666667	83	1	0,757514353	2,27254306	3	78	3	0	2
14	src/Symfony/Componen	815,4166667	274,9444444	824,8333333	48	20,66666667	62	1	0,693611638	2,080834915	3	39	3	0	2

Figura 33 – Parte do resultado das métricas para os arquivos calculadas pela classe `FileBetweenessDegreeClosenessServices`

Fonte: Autoria própria

4 CONCLUSÃO

Diante da enorme quantidade de dados gerada em um ambiente de desenvolvimento de software e das diversas oportunidades de pesquisa que surgem a partir da mineração de dados na engenharia de software, é fundamental a existência de ferramentas que proporcionem análises destes dados para um melhor entendimento da real situação do andamento dos diversos projetos hospedados nos repositórios de software.

A ferramenta apresentada neste trabalho foi criada para possibilitar a coleta de dados do `GitHub` e o desenvolvimento de algoritmos para geração de redes e cálculo de métricas sociais, técnicas e sócio-técnicas. Mostramos como é possível adicionar as métricas e redes replicando o trabalho ***Predicting Failures with Developer Networks and Social Network Analysis*** (MENEELY *et al*., 2008).

A contribuição deste trabalho encontra-se na implementação de um conjunto de classes que facilita a implementação destas métricas e redes, permitindo que vários estudos possam ser realizados com a infraestrutura de `Services` implementada.

Por outro lado, como limitação, esta ferramenta não se preocupou com aspectos visuais da representação das redes, o que torna difícil uma inspeção visual dos resultados. No entanto, tratou-se esta limitação oferecendo uma forma de exportação das redes criadas para ferramentas de redes sociais bem conhecidas.

Como trabalhos futuros, pretende-se implementar no módulo de coleta dos dados, formas de coletar informações de diversos repositórios de software, e não somente do `GitHub`. Também será implementado um módulo para visualização gráfica dos resultados, assim como existente em alguns dos trabalhos relacionados. Outra funcionalidade desejável seria o ajuste da arquitetura para permitir que novas métricas e redes sejam adicionadas em forma de *plug-ins*, por meio da importação de arquivos JAR, sem a necessidade de alteração do código.

REFERÊNCIAS

BASTIAN, M.; HEYMANN, S.; JACOMY, M. **Gephi: An open source software for exploring and manipulating networks**. In International AAAI Conference on Weblogs and Social Media, 2009.

BIRD, Christian. **Sociotechnical coordination and collaboration in open source software**. In International Conference on Software Maintenance. ICSM 2011: 568-573, Redmond, WA, EUA, 2011.

BUSE, Raymond P.L. Buse; ZIMMERMANN, Thomas. **Analytics for Software Development**, **In: Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research (FoSER 2010)**, Santa Fe, Novo México, EUA, 2010.

CEPEDA, R. S. V.; MAGDALENO, A. M.; MURTA, L. G. P.; WERNER, C. M. L. **EvolTrack: improving design evolution awareness in software development.** J. Braz. Comp. Soc. 16(2): 117-131, 2010.

COSTA, J. M. R.; Santana, F. W.; SOUZA, C. R. B. **Analisando a Evolução de Desenvolvedores de Software Livre utilizando a TransFlow.** In: Simpósio Brasileiro de Engenharia de Software, 2009, Fortaleza., 2009. p. 1-16.

DABBISH, Laura; STUART, Colleen; TSAY, Jason; HERBSLEB, James D.. **Social coding in GitHub: transparency and collaboration in an open software repository**, In: Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work, February 11-15, 2012, Seattle, Washington, EUA, 2012.

DOURISH, P.; BELLOTTI, V. **Awareness and Coordination in Shared Workspaces.** In: Conference on Computer-Supported Cooperative Work CSCW'92 (Toronto, Ontario), p. 107-114. New York: ACM., 1992.

GODFREY, Michael W.; HASSAN, Ahmed E.; HERBSLEB, James D.; MURPHY, Gail C.; ROBILLARD, Martin P.; DEVANBU, Premkumar T.; MOCKUS, Audris; PERRY, Dewayne E.; NOTKIN, David. **Future of Mining Software Archives: A Roundtable**, IEEE Software 26(1): 67-70 (2009), 2009.

HALL, Mark; FRANK, Eibe; HOLMES, Geoffrey; Bernhard; P PFAHRINGER, Peter R.; WITTEN, Ian H.. **The WEKA data mining software: an update**. SIGKDD Explorations, 11(1):10-18, 2009.

HASSAN, Ahmed E. **The Road Ahead for Mining Software Repositories**, In: Proceedings of the Future of Software Maintenance (FoSM), Beijing, China, 2008.

HASSAN, Ahmed E.; XIE, Tao. **Software Intelligence: Future of Mining Software Engineering Data,** In: Proceedings of FSE/SDP Workshop on the Future of Software Engineering Research (FoSER 2010), Santa Fe, Novo México, EUA, 2010.

HORA, A.; ANQUETIL, N.; DUCASSE, S.; BHATTI, Muhammad U.; COUTO, C.; VALENTE, Marco T.; MARTINS, J. **Bug Maps: A Tool for the Visual Exploration and Analysis of Bugs**. CSMR 2012: 523-526, 2012.

MAGDALENO, A. M.; WERNER, C. M. L.; ARAUJO, R.M. **Estudo de Ferramentas de Mineração, Visualização e Análise de Redes Sociais**, Relatório Técnico ES-735, PESC-COPPE,Rio de Janeiro, 2010

MENEELY, A.; WILLIAMS, L.; SNIPES, W.; OSBORNE, Jason. **Predicting failures with developer networks and social network analysis**. In: 16th ACM SIGSOFT International Symposium on Foundations of software engineering
Pages 13-23 ACM New York, NY, USA. 2008.

NOOY, W.; BATAGELJ, V.; MRVAR, A. **Exploratory Social Network Analysis with Pajek**, CUP, January 2005; ESNA page. P. Doreian, V. Batagelj, A. Ferligoj: Generalized Blockmodeling, CUP, 2004.

SANTANA, F. W.; OLIVA, G. A ; DE SOUZA, C. R. B. ; GEROSA, M. A. **XFlow: An Extensible Tool for Empirical Analysis of Software Systems Evolution.** In: VIII Experimental Software Engineering Latin American Workshop, 2011, Rio de Janeiro. Anais do VIII Experimental Software Engineering Latin American Workshop. Rio de Janeiro: PUC-Rio, 2011. p. 1-12.

SANTOS, M. B.; DE SOUZA, C. R. B. **Visualização Temporal de Redes Sociais com o OSSNetwork.** In: Workshop on Information Visualization and Analysis in Social Networks, 2008, Campinas., 2008. p. 79-88.

SARMA, A.; MACCHERONE, L.; WAGSTROM, P.; HERBSLEB, J. **Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development**, Proceedings of the Thirty-first International Conference on Software Engineering, Vancouver, Canada, 2009.

XIE, T.; HASSAN, Ahmed E. **Mining Software Engineering Data.** Presented at the

31st International Conference on Software Engineering (ICSE 2009), Tutorials, Vancouver, Canada, 2009.

XIE, T.; HASSAN, Ahmed E. **Mining Software Engineering Data.** Presented in the 33rd International Conference on Software Engineering (ICSE 2011), Technical Briefing, Honolulu, Hawaii, 2011.