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

EDUARDO ROBERTO GRECO

AGRUPAMENTO DE PROJETOS SIMILARES UTILIZANDO MÉTRICAS E ALGORITMO BSAS.

TRABALHO DE CONCLUSÃO DE CURSO

CAMPO MOURÃO - PR

2015

EDUARDO ROBERTO GRECO

AGRUPAMENTO DE PROJETOS SIMILARES UTILIZANDO MÉTRICAS E ALGORITMO BSAS.

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de Tecnólogo em Tecnologia em Sistemas para Internet.

Orientador: Prof. Me. Igor Scaliante Wiese

Coorientador: Prof. Dr. Reginaldo Ré

CAMPO MOURÃO - PR

2015

Dedico este trabalho de conclusão de curso aos meus pais, Roberto Greco e Vânia Peterlini Greco; ao orientador pela ajuda no decorrer do trabalho; aos professores e amigos.

AGRADECIMENTOS

Gostaria primeiramente de agradecer a Deus por ter me dado saúde e persistência para superar todos os obstáculos e sempre iluminar o meu caminho.

Aos meus pais Roberto Greco e Vânia Peterlini Greco, ao meu irmão João Vitor Greco, e toda minha família que sempre estiveram presentes, compreenderam as minhas ausências nos momentos de estudo e nunca mediram esforços para que eu pudesse alcançar este degrau da minha vida, pelo carinho, incentivo e amor.

Agradeço a todos os professores que me conduziram durante a graduação, em especial ao meu orientador Prof. Me. Igor Scaliante Wiese, por toda a paciência, dedicação e confiança em mim depositadas, fatores únicos esses que, possibilitaram a realização e conclusão deste trabalho.

Agradeço ao amigo Douglas Nassif Roma Junior pela parceria e ajuda no desenvolvimento do trabalho.

E a todos que direta ou indiretamente fizeram parte da minha formação e conquista, os meus sinceros agradecimentos.

ATA DA DEFESA DO TABALHO DE CONCLUSÃO DE CURSO

Às **dezenove e trinta minutos** do dia **quatorze de julho de 2015** foi realizada no laboratório E-103 a sessão pública da defesa do Trabalho de Conclusão do Curso Superior de Tecnologia em Sistemas para Internet do acadêmico **EDUARDO ROBERTO GRECO** com o título **AGRUPAMENTO DE PROJETOS SIMILARES UTILIZANDO MÉTRICAS E ALGORITMO BSAS**. Estavam presentes, além do acadêmico, os membros da banca examinadora composta pelo professor **Igor Scaliante Wiese** (Orientador-Presidente), pelo professor **Marco Aurélio Graciotto Silva** e pelo professor **Diego Bertolini Gonçalves**. 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 **8,5** (**oito e meio**). 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:

- FAZER AS CORREÇÕES DA BANCA

Campo Mourão, 14 de julho de 2015.

A folha de aprovação assinada encontra-se na coordenação do curso

RESUMO

GRECO, Eduardo Roberto. AGRUPAMENTO DE PROJETOS SIMILARES UTILIZANDO MÉTRICAS E ALGORITMO BSAS.. 39 f. Trabalho de Conclusão de Curso – Curso de Graduação de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Campo Mourão - PR, 2015.

Quando pesquisadores vão realizar estudos empíricos, uma das primeiras tarefas de um método de pesquisa empírica consiste na seleção de quais projetos serão utilizados para realização do estudo. Pesquisadores desejam escolher projetos similares, para tentar indicar uma possível generalização dos seus resultados para outros contextos. Para cada tipo diferente de estudo, pode ser necessário um conjunto diferente de projetos, com características específicas. Para resolver esse problema, existem métricas e algoritmos de agrupamento. Neste trabalho foi desenvolvida uma ferramenta que permite aos pesquisadores realizar a mineração de dados de projetos de software hospedados no GitHub e selecionar métricas de *pull request*, do perfil do usuário, do *log* do *commit* e do projeto para realizar o cálculo com os resultados obtidos nos projetos coletados, além de executar o algoritmo BSAS para agrupar projetos similares baseado nas suas características.

Palavras-chave: Mineração de dados, Métricas, Agrupamento de projetos

ABSTRACT

GRECO, Eduardo Roberto. AGRUPAMENTO DE PROJETOS SIMILARES UTILIZANDO MÉTRICAS E ALGORITMO BSAS.. 39 f. Trabalho de Conclusão de Curso – Curso de Graduação de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Campo Mourão - PR, 2015.

When researchers will conduct empirical studies, one of the first tasks of an empirical research method consists of selecting which projects will be used to conduct the study. Researchers want to choose similar projects, to try to indicate a possible generalization of the results to other contexts. For each different type of study, a different set of projects may be necessary, with specific characteristics. To solve this problem there are metrics and clustering algorithms. In this work it was developed a tool that allows researchers to conduct mining software project data hosted on GitHub and select metrics pull request, user profile, log commit and project to perform the calculation with the results listed in the project, and running the BSAS algorithm to group similar projects based on their characteristics.

Keywords: Data Mining, Metrics, Project grouping

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A mineração de repositórios 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* (HASSAN; XIE, 2010). Nesse contexto, a quantidade de estudos empíricos que envolvem seleção de projeto e a necessidade de saber o quanto os resultados das pesquisas são generalizáveis entre contextos tem ganhado importância (NAGAPPAN et al., 2013).

Assim, quando pesquisadores vão realizar estudos empíricos, uma das primeiras tarefas de um método de pesquisa empírica consiste na seleção de quais projetos serão utilizados para realização do estudo. Para cada tipo diferente de estudo, os pesquisadores podem necessitar de um conjunto diferente de projetos, com características específicas. Isso é importante pra saber o quanto seus resultados podem ser generalizados para outros contextos (ZIMMERMANN et al., 2009).

Por exemplo, para determinado estudo, pode ser importante encontrar projetos similares considerando a quantidade de *commits* diários, o número de comentários realizados e a quantidade de arquivos modificados em média por *commit*. No entanto, para outros estudos é importante saber a média de defeitos por arquivo, a quantidade de testes por classe e a quantidade de linhas adicionadas e removidas em cada *commit*. Esta escolha muitas vezes não é óbvia (ZIMMERMANN et al., 2009; NAGAPPAN et al., 2013).

Dado a dificuldade de selecionar projetos, uma das alternativas consiste em verificar se os projetos escolhidos para uma análise são similares ou diferentes, e além disso, verificar em quais contextos eles podem ser considerados similares ou diferentes. Se os projetos selecionados realmente representam toda uma amostra ou não.

Três estudos destacam-se quando similaridade dos projetos são observados. Nagappan et al. (2013) utilizou os dados de projetos de *software* hospedados no site Ohloh... (2015) para calcular similaridade entre os projetos. O Estudo considerou os conceitos de representatividade e cobertura e técnicas de estatística para determinar quão similar os projetos são. Neste estudo

os autores utilizam medidas como total de linhas de código, número de desenvolvedores, linguagem principal de programação, domínio de projeto, atividade recente, idade do projeto, e muitas outras, mas não adicionam nenhuma característica social do desenvolvimento do *software*. Zimmermann et al. (2009) criou um conjunto de diretrizes e regras para prever defeitos entre projetos. No seu estudo, utilizou medidas como a quantidade de linhas adicionadas, excluídas e alteradas. Apresentou um exemplo de uso para prever defeitos medindo a similaridade entre os navegadores Firefox e Internet Explorer. Jureczko e Madeyski (2010) realizou agrupamentos em projetos de software, utilizou agrupamentos hierárquicos com algoritmo K-means e também rede neural de Kohonen's para encontrar grupos de projetos semelhantes.

Nesse contexto, o objetivo deste trabalho consiste em oferecer uma ferramenta para encontrar similaridade entre projetos hospedados no GitHub. O GitHub foi escolhido pela quantidade de projetos disponíveis e pela diversidade de informações relativas aos projetos, as tarefas, as modificações dos artefatos e também ao perfil dos usuários que contribuem com os projetos. Este trabalho estende a ferramenta JGitMinerWeb, adicionando os aspectos de perfil de usuário na coleta de dados, a implementação de métricas relacionadas às categorias identificadas na literatura e opção para agrupar projetos similares utilizando o algoritmo BSAS. Por fim, é realizada uma análise preliminar exploratória dos dados obtidos com cada conjunto de métricas: Métricas de *pull request*, do perfil do usuário, do *log* do *commit* e do projeto. Nesse trabalho, é denominado métrica a medição das características/atributos do software.

Esse trabalho esta estruturado da seguinte forma. O Capítulo 2 apresenta a fundamentação teórica, o GitHub, as métricas implementadas, as medidas de similaridade e agrupamento. O Capítulo 3 apresenta o estado da arte. O Capítulo 4 apresenta o objetivo, os objetivos específicos, a ferramenta JGitMinerWeb estendida neste trabalho, os métodos utilizados e os resultados obtidos. Por fim, o Capítulo 5 apresenta as conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 GITHUB

O termo "repositório de *software*" compreende o conjunto de ferramentas utilizadas pelos desenvolvedores que permitem o armazenamento de diferentes informações sobre a evolução do desenvolvimento do *software*. Por exemplo, um repositório de *software* pode armazenar os artefatos criados e modificados durante o desenvolvimento (por exemplo, código fonte), as tarefas relatadas pelos usuários e a troca de mensagens dos desenvolvedores em listas de e-mail e nas próprias tarefas.

Muitos desses repositórios tem oferecido esses serviços na *web*, possibilitando que desenvolvedores distribuídos globalmente possam colaborar para a evolução e manutenção de *software*. O GitHub é um exemplo de repositório, que especialmente oferece recursos para aumentar as interações sociais entre os desenvolvedores, uma vez que o desenvolvimento do *software* tem natureza sócio-técnica.

Dentre os serviços oferecidos no GitHub destacam-se a possibilidade dos desenvolvedores submeterem suas contribuições de código para o projeto (*pull request*), visualizarem e revisarem código enviado por outros desenvolvedores e trocarem mensagens nas tarefas (denominadas *issues*). Considerando o aspecto social, cada desenvolvedor pode criar o seu perfil, seguir projetos e seguir outros desenvolvedores. Cada projeto hospedado no Github pode ser público ou privado. O controle de versão dos artefatos modificados é feito pelo sistema de controle de versão Git.

Neste trabalho os dados dos repositórios de cada projeto, do perfil dos usuários, das *issues e pull requests* e das modificações dos arquivos foram coletados para comparar a similaridade entre os projetos. Mais detalhes são mostrados na Seção 4.3. A obtenção de dados do GitHub permite aos pesquisadores analisar informações técnicas e sociais a respeito da evolução de cada projeto, podendo comparar projetos e realizar uma série de estudos empíricos com estes dados.

2.1.1 GITHUB API

Uma biblioteca chamada GitHub API[1] é oferecida pelos próprios mantenedores do repositório. A biblioteca atualmente encontra-se na versão v3, é escrita em JAVA e sua comunicação é via JSON com o GitHub. Na biblioteca estão mapeadas as entidades do GitHub em forma de classes que permitem a recuperação dos dados referente ao projeto, *pull request*, perfil do usuário, *log* do *commit* dos *software* hospedados no repositório.

Para coletar um repositório utilizando a API do GitHub é relativamente simples. É necessário realizar autenticação pela API e executar a chamada de um serviço utilizando o parâmetro com o nome do repositório que deseja ser coletado, e a mesma retorna um objeto com o resultado.

2.2 MÉTRICAS

Uma métrica é a medição de um atributo (propriedades ou características) de uma determinada entidade (produto, processo ou recursos) (KOSCIANSKI; SOARES, 2007). Esta seção apresenta as métricas implementadas na ferramenta JGitMinerWeb 2.0. Mais detalhes da implementação estão na Seção 4.3 deste trabalho. As métricas obtidas foram categorizadas e foram selecionadas a partir de trabalhos como Tsay et al. (2014), Marlow et al. (2013), Dabbish et al. (2012) e Khadke et al. (2012), que utilizaram o GitHub como fonte de dados em suas pesquisa.

A partir da leitura desses trabalhos, as métricas foram classificadas em quatro grupos: métricas de *pull request*, métricas do perfil do usuário, métricas do *log* do *commit* e métricas do projeto. Esses quatro grupos representam métricas de diferentes níveis de abstração que podem ser observados no Github. As subseções abaixo descrevem cada uma das métricas identificadas em cada grupo.

2.2.1 MÉTRICAS DE *PULL REQUEST*

Esta seção apresenta as métricas calculadas a partir de dados de *pull request*. O primeiro conjunto de métricas foi obtido do trabalho de Tsay et al. (2014). O autor utiliza as métricas de teste de inclusão, tamanho do *commit*, número de arquivos alterados e distância social.

O **teste de inclusão** é medida como uma variável dicotômica (sim ou não) que indica

[1] https://developer.github.com/v3/

se o *pull request* possui ou não casos de teste adicionados juntos com o código. O cálculo dessa métrica é realizado observando os caminhos de arquivo em cada *pull request*, procurando a palavra *test* (em inglês) no caminho dos arquivos que foram modificados em um *pull request*. A verificação observa a ocorrência da palavra *test* como parte do caminho, por exemplo, "projeto/test/..."ou os nomes dos arquivos "projeto/tela/TestTela.java".

Tamanho do *commit* indica o número de linhas alteradas em *pull request*. Esta métrica é utilizada para alertar desenvolvedores sobre grandes modificações em uma única tarefa. Ela pode ser obtida diretamente pela API do GitHub que retorna a quantidade de linhas modificadas por cada arquivo alterado em um *pull request*.

O **número de arquivos alterados** representa o número de arquivos alterados no *pull request*. *Pull requests* que possuem um grande número de arquivos tendem a ser muito mais difíceis para os gerentes de projetos realizar a avaliação. Ela pode ser obtida diretamente pela API do GitHub que retorna a quantidade de linha modificada por cada arquivo alterado em um *pull request*.

A **distância social** indica se o desenvolvedor que submeteu uma contribuição em um *pull request* segue o desenvolver membro do projeto que fecha e realiza o *merge* do código submetido. O valor retornado por essa métrica é sim ou não. O cálculo é realizado em duas dimensões diferentes, na qual é uma medida de distância social e a outra para interação anterior.

Interação prévia ou anterior, indica que os membros do núcleo de um projeto, especialmente quando se tenta recrutar novos membros, utilizam contribuições anteriores como um sinal da confiabilidade de um contribuinte. É levado em consideração eventos que incluem participação em *pull request* e comentários sobre vários artefatos do GitHub. Para medir, é realizado a contagem do número de eventos que o usuário tenha participado no projeto antes de realizar um *pull request*. Esta métrica foi utilizada por Dabbish et al. (2012).

Contribuições amplamente discutidas foi utilizada por Marlow et al. (2013) para verificar o número de discussões representados pelos comentários gerados pelo *pull request*. *Pull request* difíceis de entender ou resolver tendem a requerer uma maior discussão, negociação e/ou explicação. *Pull request* com muitos comentários também tendem a sinalizar controvérsia. Para medir o nível de discussão, é realizada a contagem do número de comentários em *pull request* que já foram fechados.

Total de colaboradores que comentaram uma *issue* foi utilizada por Tsay et al. (2014) para verificar o número de discussões por colaboradores do projeto em uma determinada *issue*.

2.2.2 MÉTRICAS DO PERFIL DO USUÁRIO

Esta seção apresenta as métricas calculadas a partir de dados do perfil do usuário. Na página do perfil de um usuário GitHub encontram-se dados que possibilitam o cálculo de métricas.

Número de seguidores que um usuário possui é usada como um sinal de status dentro da comunidade. Por exemplo, os usuários com muitos seguidores são tratados como celebridades locais. O cálculo é realizado contando a quantidade de seguidores(*followers*) no perfil de um usuário (TSAY et al., 2014).

O **status do colaborador** é utilizada para verificar o status de cada colaborador do projeto. É definida por Tsay et al. (2014) como uma variável dicotômica (sim/não) para o status de usuário colaborador.

2.2.3 MÉTRICAS DO *LOG* DO *COMMIT*

Esta seção apresenta as métricas calculadas após o *commit*. Por exemplo Khadke et al. (2012) mostra medidas como número de linhas alteradas.

A métrica de **Linhas alteradas** verifica a quantidade de linhas alteradas do código fonte. É realizado o cálculo após o *commit* para verificar se possui mais ou menos linhas.

Número de *commit* indica a atividade do projeto em termos de *commits* realizados. É usada para calcular o número de *commit* de um projeto somando-se os *commits* feitos em um intervalo de tempo (KHADKE et al., 2012).

Total de colaboradores que fazem o *commit* dos arquivos é utilizada para indicar a colaboração de desenvolvedores do projeto. É usada para calcular o número de colaboradores distintos do projeto em um determinado período de tempo (TSAY et al., 2014).

2.2.4 MÉTRICAS DO PROJETO

Esta seção apresenta as métricas calculadas a partir de dados dos projetos armazenados no GitHub. Em um projeto do GitHub encontram-se diversos dados que possibilitam o cálculo de métricas.

Stars (Estrelas) indica o numero de estrelas de um projeto. O número de estrelas como um sinal de interesse da comunidade com o projeto. Quanto maior a quantidade de estrelas em um projeto, significa que o mesmo possui constantes contribuições e boa reputação. As estrelas

são dadas aos projetos pelos desenvolvedores. (TSAY et al., 2014).

Total de Colaboradores de um projeto Segundo Tsay et al. (2014) é usado o número de colaboradores como um proxy para o tamanho relativo da equipe de desenvolvimento envolvido em um projeto GitHub particular.

Idade do repositório usada como um indicador da maturidade do repositório, esta métrica representa a idade do projeto. Ela é calculada para obter a quantidade de tempo em meses que um projeto existe no Github. O cálculo é realizado obtendo a diferença de meses do primeiro para o último *commit* (TSAY et al., 2014).

Número de *issues* em aberto do repositório, o cálculo verifica a quantidade de *issues* em aberto no repositório. Um repositório com muitas *issues* em aberto podem estar mais dispostos a aceitar pedidos de *pull requests* para resolver esses problemas (KHADKE et al., 2012).

2.3 MEDIDAS DE SIMILARIDADE E AGRUPAMENTO

Para comparar os projetos e encontrar similaridade entre eles, é necessário o uso de medidas de similaridade e algoritmos de agrupamento de dados. Esta seção, apresenta as técnicas utilizadas neste trabalho.

2.3.1 ALGORITMOS DE AGRUPAMENTO

O objetivo do agrupamento é encontrar grupos úteis de objetos, onde a utilidade seja definida pela análise de dados. Há diversas noções diferentes de um grupo que podem ser úteis na prática (TAN et al., 2009). Esta subseção apresenta os algoritmos de agrupamento **BSAS** (*Basic sequential algorithmic scheme*) e **K-means**.

BSAS é um algoritmo de agrupamento básico que utiliza métodos simples e rápidos para formar agrupamentos - *clusters*. *Cluster* é um conjunto de objetos no qual cada objeto está mais próximo (ou é mais similar) a objetos dentro do *cluster* do que qualquer objeto fora do *cluster* (SEGARAN, 2007).

Na entrada do algoritmo são adicionados todos os vetores de características, que vão ser classificados em grupos. Além dos vetores é adicionado como parâmetro o valor da distância de tolerância para a criação dos grupos e a quantidade máxima de grupos que podem ser criados. O valor da distância de tolerância é definido aleatoriamente e é muito importante pois tem um efeito direto sobre o número de grupos formados. Se esse valor é muito pequeno

grupos desnecessários são criados e se for muito grande é criado um número menor de grupos necessários (KAINULAINEN; JUKKA, 2002).

Para poder agrupar conjuntos de objetos é necessário medir a similaridade entre eles. Um dos métodos utilizado pelo algoritmo é uma função de similaridade d(x,C) para calcular a distância entre o vetor de características x e um grupo C. Neste trabalho a função utilizada foi a distância euclidiana, pois é uma distância geométrica no espaço multidimensional. Tendo-se $X = [X_1, X_2, ..., X_p]$ e $Y = [Y_1, Y_2, ..., Yp]$, a distância entre os pontos é definida por:

$$d_(xy) = \sqrt{(X_1 - Y_1)^2 \quad + (X_1 - Y_1)^2 \quad + ...(X_p - Y_p)^2 \quad} = \sqrt{\sum_{i=1}^{p} (X_i - Y_i)^2}$$

Exemplo de como realizar o cálculo da distância entre os elementos $X_0 = (1,2)$ e $X_1 = (3,4)$.

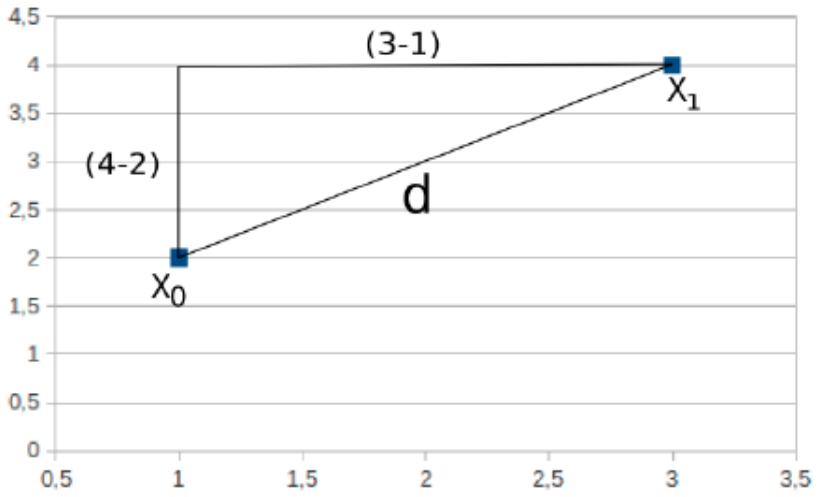

Figura 1: Exemplo de um espaço euclidiano

Fonte: (ZAÏANE et al., 2002)

O resultado do cálculo da distância euclidiana entre o elemento X_0 e X_1 é:

$$dx_0x_1 = \sqrt{(3-1)^2 \quad + (4-2)^2 \quad} = \sqrt{8} = 2,83$$

BSAS cria os grupos conforme a necessidade de novos agrupamentos. Esse novo grupo somente é criado quando o valor da menor distância entre os projetos é maior que o da distância de tolerância informado, e quando a quantidade total de grupos de um agrupamento é superior ao valor estipulado no parâmetro de entrada. Quando adicionado o vetor em um grupo existente ele recalcula o valor do centróide (Ponto central de um grupo) que é igual a soma das distâncias dividido pela quantidade de vetores do grupo. Também é recalculada a distância euclidiana entre os vetores desse mesmo grupo (KAINULAINEN; JUKKA, 2002).

A ideia básica do algoritmo é atribuir um vetor de característica a um grupo existente ou para um novo, conforme citado acima. Os resultados por exemplo podem ser *N* grupos com *N* vetores.

Nesse trabalho foi utilizado o algoritmo BSAS para gerar o agrupamento dos projetos similares. O algoritmo BSAS permite utilizar uma função de similaridade, neste trabalho utilizada a euclidiana, também atribuí automaticamente o valor do centróide (TAN et al., 2009). Existem diversos algoritmos de agrupamento, como por exemplo o K-means, segundo TAN et al. (2009), é um dos algoritmos de agrupamento mais antigos. Abaixo é apresentado alguns problemas de K-means que não são encontrados em BSAS.

K-means utiliza uma técnica de agrupamento baseada em centróides que criam um particionamento de um nível dos objetos de dados. Esse valor inicial do centróide deve ser informado manualmente. Primeiro é escolhido *K* centróides iniciais, onde *K* é um parâmetro para saber o número de grupos desejados. Cada ponto é atribuído ao centróide mais próximo, e a coleção de pontos atribuída ao centróide é um grupo. Então a cada ponto atribuído ao grupo, o centróide daquele grupo é atualizado. Os passos se repetem até que nenhum ponto mude de grupo, até que os centróides permaneçam os mesmos (TAN et al., 2009).

A definição dos centróides iniciais é a etapa chave do procedimento K-means básico. Quando os centróides são selecionados aleatoriamente não é obtido um agrupamento ideal. Uma técnica que é utilizada para resolver esse problema é executar múltiplas vezes, cada uma com um conjunto diferente de centróides escolhidos aleatoriamente e depois selecionar o conjunto de grupos com a SSE (Soma do erro quadrado) mínima, na qual é a qualidade de um agrupamento. Porém essa técnica pode custar muito tempo, pois é demorada e ainda a estratégia pode não funcionar perfeitamente dependendo do conjunto de dados e do número de grupos procurados. Outra técnica bastante utilizada é usar uma amostra de pontos e agrupá-los utilizando agrupamento hierárquico. K grupos são extraídos do agrupamento hierárquico e os centróides desses grupos são usados como centróides iniciais. Essa abordagem funciona bem se a amostra for relativamente pequena e K for relativamente pequeno comparado com o tamanho da amostra.

Uma outra abordagem para resolver o problema de seleção de centróides é selecionar o primeiro ponto aleatoriamente ou pegar o centróide de todos os pontos. Para cada centróide inicial sucessivo, deve selecionar o ponto que estiver mais distante de qualquer um dos centróides já selecionados. Desta forma, obtêm um conjunto inicial que seja selecionado aleatoriamente e bem separado. Infelizmente, essa abordagem pode selecionar elementos externos, além de ser custoso calcular o ponto mais distante do conjunto de centróides iniciais (TAN et al., 2009).

3 ESTADO DA ARTE

3.1 TRABALHOS RELACIONADOS

O trabalho de Nagappan et al. (2013) realizou um estudo para encontrar projetos similares entre os projetos de *software* hospedados no site Ohloh... (2015). As medidas de representatividade, diversidade e cobertura foram utilizadas como medidas de similaridade e agrupamento dos projetos. De cada projeto hospedado no Ohloh... (2015) foram obtidas as seguintes informações: linguagem principal de programação, o total de linhas de código, número de colaboradores, número de linhas modificadas, número de confirmações, idade do projeto e atividades do projeto.

Nagappan et al. (2013) definiu funções de similaridade que decidem se dois projetos são semelhantes. Essas funções podem ser configuradas de diferentes formas para diferentes tópicos de pesquisa e áreas. O mesmo apresenta um exemplo sobre cobertura e seleção de projeto e utiliza algoritmos implementados[1] em linguagem R.

Para a coleta de dados os seguintes passos foram utilizados: 1) Extraiu os identificadores dos projetos ativos usando a API do Ohloh. 2) Para cada identificador de projeto, foram extraídos três diferentes. O primeiro é a categoria de análise que possui dados sobre a principal linguagem de programação, o tamanho do código-fonte e os contribuidores. A segunda é a categoria de atividades que resume o quanto desenvolvedores alteram o código-fonte a cada mês. Por fim, coletou o que é chamado de categoria *Factoid*. Esta categoria contém observações básicas sobre projetos como o tamanho da equipe, a idade do projeto, relação de comentários, e conflitos de licença. 3) Agregou os arquivos XML retornados pela API do Ohloh e converteu em arquivos de texto separados por tabulação usando um script personalizado.

Dois estudos de caso foram apresentados. No primeiro estudo de caso os autores apresentam uma técnica geral para avaliar a cobertura de uma amostra em relação a uma população de projetos de *software* e selecionam uma amostra com cobertura máxima. Em seguida, os autores apresentam uma técnica por meio do cálculo da cobertura de pesquisa na ICSE (*Inter-*

[1] http://sailhome.cs.queensu.ca/replication/representativeness/

national Conference on Software engineering) e FSE (*Foundations of Software Engineering*) fornecendo os meios adequados para cobertura de eventos e seleção de projetos em geral.

No primeiro experimento Nagappan et al. (2013) contou o conjunto de projetos necessários para cobrir toda a população de projetos do Ohloh. Os primeiros 50 projetos cobriu 15,3% do universo, 392 projetos cobriu 50%, e 5.030 projetos abrangeram 100% do universo. Segundo os resultados obtidos, os pequenos projetos de *software* escrito em linguagens dinâmicas dominam a lista (sete dos nove primeiros estão em Ruby ou Python e sob 2000 *LOC*). Mesmo considerando todos os 15 projetos, esses projetos em conjunto representam menos de 200.000 *LOC* e pouco mais de mil *commits*, uma ordem de grandeza menor do que para o Apache HTTP, Mozilla Firefox, ou Eclipse JDT que são projetos maiores.

O tempo e o espaço necessário para analisar ou avaliar esses projetos são bastante baixos, proporcionando uma oportunidade propícia para pesquisadores conseguir impacto sem grandes demandas de recursos. Esse resultado também contraria uma crítica comum de algumas pesquisas de engenharia de *software*, no qual citam que a investigação deve sempre ter uma escala para *software* maior e prestar menos atenção em projetos de menor dimensão (NAGAPPAN et al., 2013).

No segundo estudo Nagappan et al. (2013) aplicou sua técnica instanciado com o universo Ohloh para pesquisas no campo de engenharia de *software* realizadas ao longo de dois anos na ICSE (*International Conference on Software engineering*) e FSE (*Foundations of Software Engineering*). Para criar o conjunto de dados teve como base os projetos de *software* que foram analisados e registrados o número e os nomes dos projetos em uma planilha. Em seguida, verificou para cada um dos projetos de *software* no Ohloh seu identificador correspondente, que utilizou para comparar os dados.

A análise das conferências ICSE e FSE revelou vários estudos de grande escala que analisaram centenas, se não milhares de projetos. Nagappan et al. (2013) encontrou 635 projetos originais que foram analisados pelo ICSE e FSE no período de dois anos. Destes, mapeou 207 projetos ativos para o universo Ohloh. Os projetos mais estudados foram as ferramentas de Eclipse Java de Desenvolvimento (JDT) em 16 artigos, Apache HTTP Server em 12 artigos, gzip, jEdit, Apache Xalan C ++, e Apache Lucene em cada oito artigos e Mozilla Firefox em 7 artigos. Outro projeto frequentemente estudado é o Linux, que foi analisado em 12 artigos. Os resultados permitem avaliar, quais dimensões foram boas e quais dimensões precisam ser melhoradas.

Para prever defeitos entre projetos, o trabalho de Zimmermann et al. (2009) criou um conjunto de diretrizes e regras que são descritas abaixo:

- Descreveu cada projeto com um conjunto de características. As características de um projeto é um vetor de valores. No total usou 40 características, que descreveram o domínio, processos e dados de cada projeto;

- Comparou todos os projetos em pares no que diz respeito às suas características. Para cada dois projetos o resultado é um vetor de similaridade, que descreve se as características são iguais ou diferentes. No total usou 622 vetores de similaridade;

- Para cada uma das características e os seus níveis no vetor de similaridade, realizou um teste para verificar se o nível afeta a precisão e sensibilidade. Para as 40 características, possui 125 níveis nos vetores de similaridade. Para cada nível, realizou três testes para avaliar efeito sobre a precisão, sensibilidade (em um total de 375 testes);

- Criou árvores de decisão a partir dos dados de similaridade para descrever precisão e sensibilidade. Enquanto a etapa anterior analisa os níveis de similaridade de forma independente, uma árvore de decisão também analisa as interações;

Um bom exemplo utilizado por Zimmermann et al. (2009) seria comparar o Firefox com o Internet Explorer(IE), considerando apenas um subconjunto de características. Ambos os projetos são navegadores, assim, a característica Domínio é definida como igual no vetor de similaridade. Firefox é *open source*, o IE não, portanto, *Open source* é definido como "diferente". Ambos os projetos fazem revisões de código "igual". IE é menor do que o Firefox em termos de linhas de código, portanto, "Linhas de código" é definido como "Menor".

Nesse experimento coletou métricas como a quantidade de linhas adicionados, excluídos e alteradas. Coletou a complexidade e erros *"bugs"* em pré-lançamentos de versão para construir modelos (por regressão logística) para prever defeitos do Firefox no Internet Explorer e vice-versa. Os resultados mostraram que os dados do Firefox pode prever defeitos do Internet Explorer com uma precisão de 76,47%. O sentido oposto não funcionou (apenas 4,12%).

Pode-se perguntar por que a previsão de defeito não é sempre bidirecional. Para o Firefox e Internet Explorer uma possível razão pode ser o descompasso entre o número de observações. Firefox tem mais arquivos do que o Internet Explorer. Construindo um modelo a partir de uma pequena população a chance de prever é maior e é provavelmente mais difícil do que no sentido inverso.

Jureczko e Madeyski (2010) realizou agrupamentos em projetos de software, a fim de identificar grupos de projetos com características semelhantes para previsão de defeitos. A existência desses grupos foi investigada com testes estatísticos e por comparação do valor médio

de eficiência de predição. Realizou agrupamentos hierárquicos com algoritmo K-means, bem como utilizou a rede neural de Kohonen's para encontrar grupos de projetos semelhantes.

Para cada um dos grupos identificadas uma análise foi conduzida de modo a distinguir se este grupo realmente existe. Dois modelos de previsão de defeitos foram criados para cada um dos grupos identificados. O primeiro foi baseado nos projetos que pertencem a um determinado grupo, e o segundo em todos os projetos. Os dois grupos identificados foram descritos e comparados os resultados obtidos. Os resultados de Jureczko e Madeyski (2010) apresentam definições de métodos formais para reutilizar modelos de previsão de defeito, identificando grupos de projetos dentro do qual pode ser utilizado o mesmo modelo de previsão de defeito. Além disso, aplicou um método de agrupamento.

Este trabalho diferencia o do Nagappan et al. (2013) e Jureczko e Madeyski (2010), pois utiliza o algoritmo BSAS, detalhado na Seção 2.3 para realizar o agrupamento de projetos similares hospedados no repositório GitHub. Nagappan et al. (2013) utilizou algoritmos implementados em linguagem R para verificar a similaridade entre projetos hospedados no repositório Ohloh e Jureczko e Madeyski (2010) utilizou K-means. GitHub é um dos repositórios de *software* que possui mais características sociais e é um dos maiores repositórios do mundo com 24 milhões de projetos e com 9,9 milhões de usuários (GITHUB. . . , 2015).

Métricas utilizadas por Nagappan et al. (2013) e Zimmermann et al. (2009) estão implementadas neste trabalho e descritas na Seção 2.2.

4 AGRUPAMENTO DE PROJETOS SIMILARES

4.1 OBJETIVO

O objetivo deste trabalho consiste em oferecer uma ferramenta para encontrar similaridade entre projetos hospedados no GitHub. A similaridade será obtida a partir das métricas encontradas na literatura e descritas na Seção 2.2.

4.2 OBJETIVOS ESPECÍFICOS

- Estender a coleta de dados adicionando os aspectos de perfil de usuário para a ferramenta JGitMinerWeb.
- Implementar as métricas relacionadas as categorias identificadas na literatura na ferramenta JGitMinerWeb.
- Agrupar projetos similares utilizando o algoritmo BSAS (*Basic sequential algorithmic scheme*).
- Realizar uma análise exploratória dos dados obtidos com cada conjunto de métrica.

4.3 A FERRAMENTA JGITMINERWEB

Esta seção apresenta a ferramenta JGitMinerWeb estendida neste trabalho. As subseções abaixo apresentam as alterações e melhorias realizadas.

4.3.1 JGITMINERWEB 1.0

Esta ferramenta possibilita a coleta de dados de repositórios de *software* do GitHub, fornecendo duas formas de análise dos dados coletados. A primeira forma possibilita a criação de diversas redes sociais, técnicas e sócio-técnicas para análise na ferramenta PAJEK.

A segunda forma permite a extração de medidas quantitativas que podem ser exportadas para ferramenta WEKA, para análise posterior.

A ferramenta é implementada na linguagem JAVA, utiliza o *framework JSF (Java Server Faces)* na interface e o servidor de aplicação *GlassFish*. O banco de dados é *PostgreSQL*, por se tratar de um gerenciador de banco de dados com licença *GPL (General Public License)*. A ferramenta contém três módulos: o módulo de coleta, que utiliza uma API 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.

O módulo de coleta (1) 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. O módulo de geração e exportação das redes (2) possibilita que sejam criados algoritmos para geração das redes em forma de matriz como entrada os dados vindos do GitHub. O módulo para cálculo e exportação de métricas (3) 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 (NAGAPPAN et al., 2013).

Figura 2: Visão geral da ferramenta

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. O cadastro ilustrado na Figura 3 é realizado informando o nome de usuário do dono do projeto e o nome do projeto hospedado no GitHub ou apenas a URL do projeto (1). A Figura 3 (2) exibe a lista de repositórios já cadastrados.

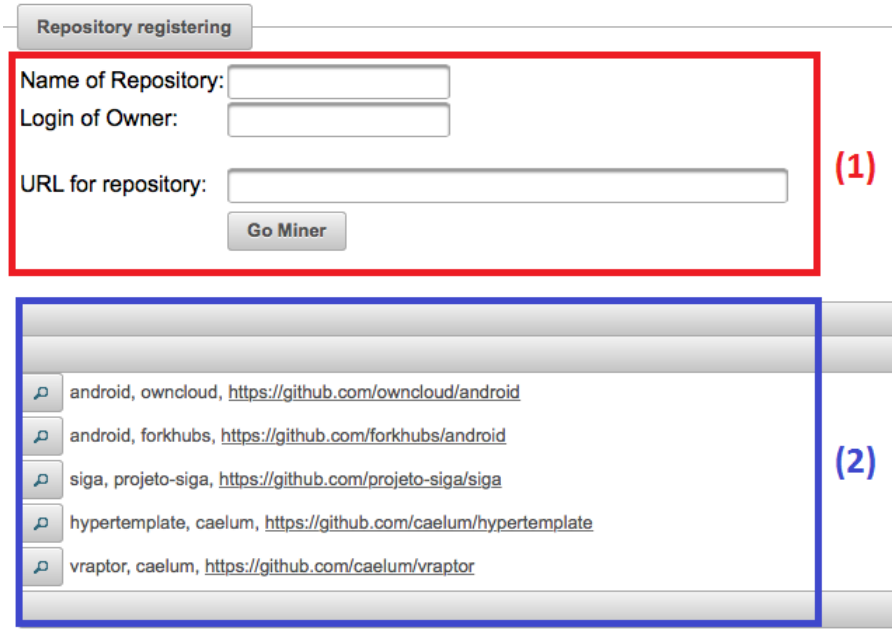

Figura 3: Página de cadastro de repositórios

Uma vez que o repositório foi cadastrado na ferramenta, ele passa a estar disponível para ser utilizado na página de coleta dos dados ilustrada na Figura 4 (1). O módulo de coleta dos dados oferece opções para o usuário escolher quais as informações que ele deseja coletar do repositório de *software* do GitHub (2). 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 calcular informações 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 (JUNIOR, 2013).

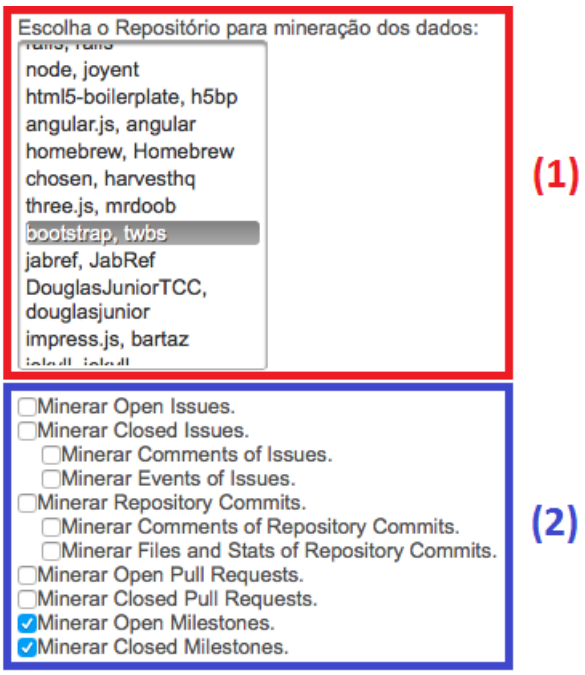

Figura 4: Página de coletados dados dos repositórios de *software* hospedados no GitHub

Após a coleta dos dados do repositório é possível gerar redes de dados. Conforme ilustrado na Figura 5. Para dar início a geração de uma rede, o usuário precisa selecionar primeiro o repositório desejado (1), em seguida informar os filtros conforme sua preferência (2). No exemplo o usuário selecionou o repositório *bootstrap*, e o filtro escolhido "usuários que modificaram o mesmo arquivo em um *pull request*". Com os dados gerados é possível realizar o download em formato de arquivo CSV dos resultados.

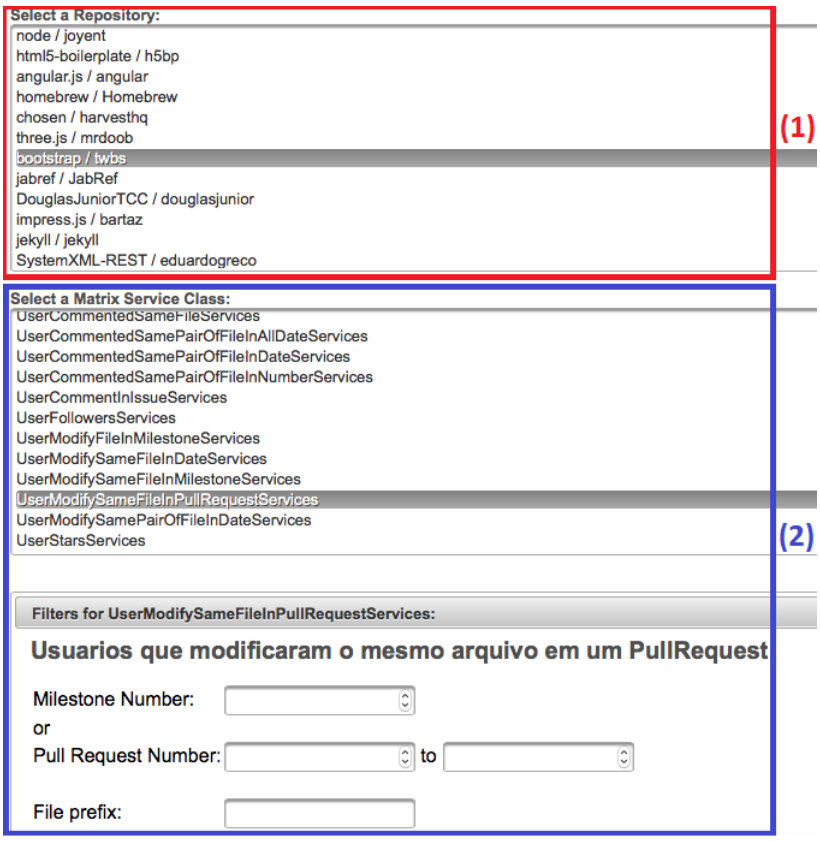

Figura 5: Página de geração das redes

4.3.2 JGITMINERWEB 2.0

A ferramenta JGitMinerWeb foi estendida neste trabalho gerando a versão 2.0. Foram adicionadas novas opções para coleta de projetos. Adicionado o cálculo de métricas. Adicionado execução do algoritmo BSAS, e adaptação no algoritmo para executar os resultados obtidos pelo cálculo das métricas. Nessa subseção estão os detalhes dessas alterações.

No módulo de coleta dos dados foram adicionadas mais opções para o usuário coletar dos projetos armazenados no repositório GitHub. Como por exemplo a mineração de *Stars*, não disponível na versão 1.0. A Figura 6 apresenta os projetos disponíveis para coleta previamente cadastrados (1), e as novas opções de dados para a coletar desses projetos disponíveis na versão 2.0 da ferramenta (2).

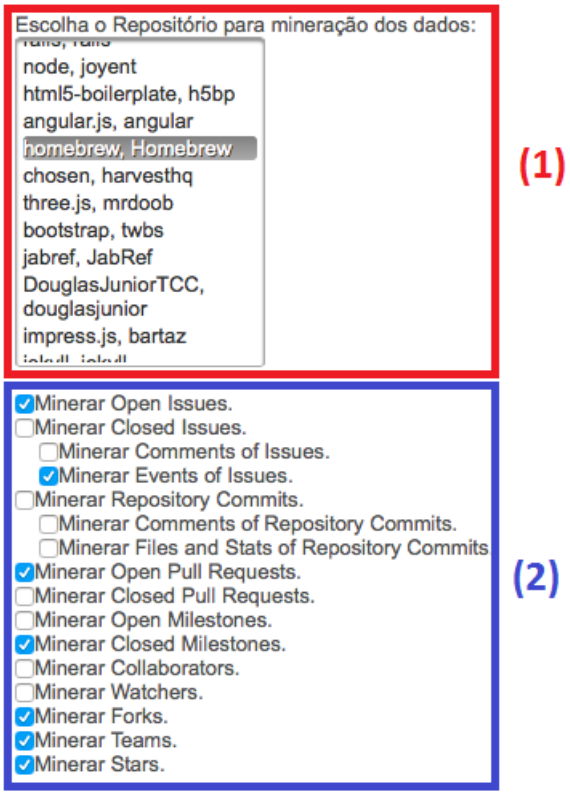

Figura 6: Nova página de coletados dados dos repositórios de *software* hospedados no GitHub

Após a coleta dos dados do repositório é possível realizar o cálculo das métricas nos repositórios selecionados. Na versão 1.0 da ferramenta é permitido apenas a seleção de um único repositório. Na versão 2.0 foi realizada uma alteração, adicionado a opção de selecionar vários repositórios. Depois de selecionar os repositórios o usuário precisa definir quais métricas ele deseja que seja realizado o cálculo. As métricas disponíveis foram descritas na Seção 2.2. A Figura 7 abaixo apresenta um exemplo de como o usuário realiza o cálculo das métricas nos repositórios. Usuário pode selecionar vários projetos (1), definir o serviço (2), selecionar as métricas e o período que deseja calcular (3).

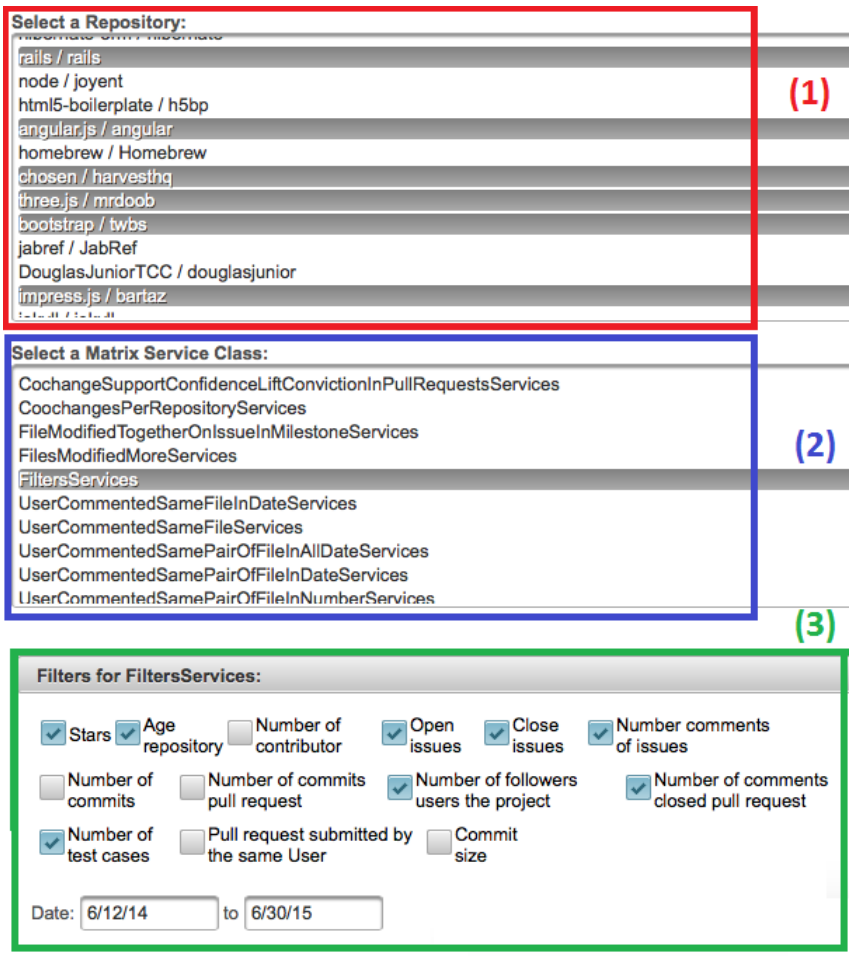

Figura 7: Seleção de projeto e cálculo de métricas

Realizado o cálculo das métricas, o usuário tem a opção de realizar o download do resultado em formato CSV ou em TXT (1), e também executar o resultado no algoritmo BSAS (2). Ambos implementados na versão 2.0 da ferramenta e ilustrado na Figura 8.

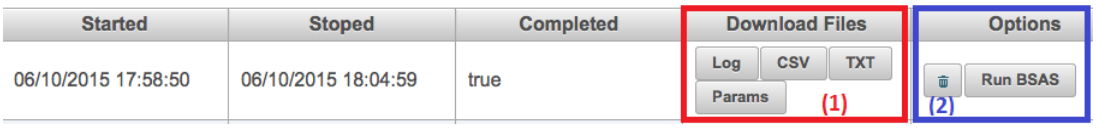

Figura 8: Opções para o resultado do cálculo das métricas

Se o usuário realizar o download do arquivo com o resultado do cálculo das métricas, na primeira coluna ele vai poder visualizar os nomes dos repositórios, e nas demais colunas os nomes das métricas calculadas. Nas linhas do arquivo são apresentados os valores obtidos. Como por exemplo a Figura 9.

	A	B	C	D	E
1	Repository_Name	Amount_Of_Stars	Repository_Age_In_Months	Amount_Of_Open_Issue	Amount_Of_Close_Issue
2	querydsl	511	46	104	1225
3	jQuery-File-Upload	15849	54	0	219
4	SlidingMenu	7745	35	269	447
5	flask	4476	62	143	1303
6	node	19868	72	1291	8410
7	homebrew	2158	72	392	38710
8	jekyll	18226	79	87	3546
9	impress.js	19718	41	187	287
10	android	588	33	249	761
11	foundation	19870	43	102	6179
12	oh-my-zsh	23427	69	594	3381
13	Font-Awesome	15640	39	2733	1107
14	android	249	24	86	38
15	express	18790	71	80	2591
16	web	910	30	35	83
17	jabref	65	15	2	35
18	html5-boilerplate	26341	64	16	1687
19	android	6515	45	148	665
20	SystemXML-REST	1	9	0	0
21	chosen	17377	49	71	0
22	hypertemplate	16	50	7	4
23	three.js	19015	62	779	5171
24	brackets	15388	42	1332	9881
25	vraptor4	202	25	27	570
26	siga	0	9	21	259

Figura 9: Resultado do cálculo das métricas

Se optar por executar o algoritmo de agrupamento de projetos usando o BSAS, o usuário precisa definir o valor da distância de tolerância para a criação dos agrupamentos, como detalhado na Seção 2.3. O resultado da execução do algoritmo BSAS é um arquivo, que na primeira coluna contém os *grupos* formados, na segunda coluna os projetos de cada *grupo*, na terceira coluna o valor da distância de cada projeto e na última coluna o valor da coesão que é a soma total da multiplicação das distâncias dos projetos do mesmo grupo. Como exemplo a Figura 10.

	A	B	C	D
1	cluster	project	distance	coesion
2	1	rails	0.0	3.57
3	1	bootstrap	4633.806102978415	3.57
4	1	angular.js	3772.448762629741	3.57
5	2	django	0.0	2.76
6	2	impress.js	3230.9358708526697	2.76
7	2	html5-boilerplate	8070.952017981676	2.76
8	2	brackets	7357.486398326707	2.76
9	2	moment	2258.4456715986107	2.76
10	2	jekyll	582.61541909093	2.76
11	2	three.js	2038.0047783463365	2.76
12	2	foundation	3304.4601830802108	2.76
13	2	chosen	3432.7848536665533	2.76
14	2	oh-my-zsh	4993.438066759465	2.76
15	2	express	860.355944689464	2.76
16	2	Font-Awesome	4146.027538688389	2.76
17	2	gitlabhq	3881.609601900029	2.76
18	2	jQuery-File-Upload	4007.2313355435945	2.76
19	2	node	5425.146055781586	2.76
20	2	d3	3263.9176003022044	2.76
21	3	guj.com.br	0.0	1.09
22	3	android	623.5581328611087	1.09
23	3	SlidingMenu	6865.120312590855	1.09
24	3	caelum-stella	656.877292433904	1.09

Figura 10: Resultado da execução do algoritmo BSAS

O algoritmo BSAS utilizado neste trabalho e a implementação das métricas está disponível em: `https://github.com/eduardogreco/DouglasJuniorTCC`.

4.4 MÉTODO

A metodologia deste trabalho está dividida em três etapas descritas abaixo:

- Coleta e critérios de comparação;
- Execução e exportação;
- Resultados;

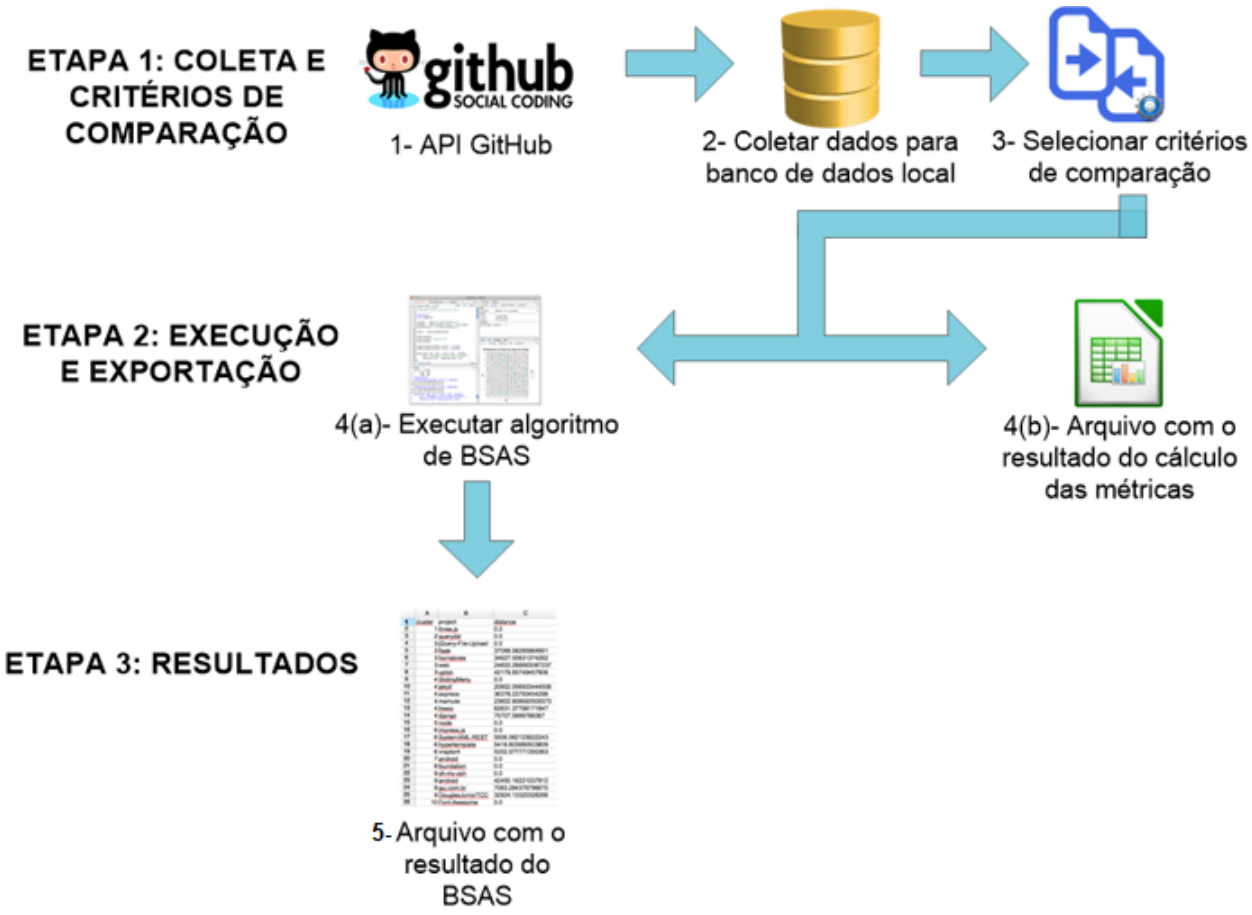

Figura 11: Etapas do projeto

A Figura 11 indica as três etapas da metodologia. A primeira etapa apresenta os processos de como realizar a coleta e a seleção dos critérios de comparação. A ferramenta JGitMinerWeb 2.0 coleta informações dos repositórios do GitHub (1). Após a coleta, os dados são armazenados em um banco de dados local (2). Consultando o banco de dados local, é possível escrever consultas que permitem a seleção de dados para realizar o cálculo das métricas atendendo aos critérios de seleção dos projetos e seleção das métricas(3).

Após o cálculo das métricas e a definição dos critérios de comparação, na segunda etapa pode-se executar o algoritmo BSAS 4(a). Antes de executar o algoritmo é necessário definir o parâmetro da distância de tolerância para a criação dos grupos. Esse valor é definido por tentativa e erro, dependendo da quantidade de grupos que deseja ser formado. O algoritmo BSAS implementado em *JAVA* foi adicionado e adaptado na ferramenta JGitMinerWeb 2.0 para realizar o agrupamento dos projetos similares de acordo com o valor obtido de cada métrica selecionada. Na segunda etapa também é disponível realizar o download de um arquivo CSV contendo o resultado obtido pelo cálculo das métricas 4(b). As linhas deste arquivo representam os projetos selecionados, e as colunas representam as métricas selecionadas pelos usuários.

Na terceira e última etapa, é gerado um arquivo com os agrupamentos realizado pelo algoritmo BSAS (5). Com base nessas informações o usuário pode escolher os projetos que deseja incluir no seu estudo.

A fim de ilustrar um exemplo de uso, consideramos por exemplo, que um pesquisador deseja agrupar projetos com características semelhantes utilizando as métricas de *pull request*. O pesquisador seleciona que deseja coletar dados das *issues* e seus comentários, deixando de lado os dados dos *commits*, *forks*, etc. No exemplo, vamos considerar que o pesquisador seleciona 10 projetos, no período entre 2008-2010 e as métricas classificadas como *pull request*. É realizado o cálculo das métricas e executado o algoritmo BSAS. A partir dos dados gerados pelo algoritmo o pesquisador pode analisar os agrupamentos gerados.

4.5 ESTUDO EXPLORATÓRIO DO AGRUPAMENTO DE PROJETOS UTILIZANDO AS MÉTRICAS DE *PULL REQUEST*, DO PERFIL DO USUÁRIO, DO *LOG* DO *COMMIT* E DO PROJETO

Esta seção apresenta os resultados obtidos nesse trabalho na execução do algoritmo de agrupamento BSAS, como entrada o resultado do cálculo das métricas em 50 projetos mais populares[1] do GitHub. Esses projetos são aqueles que possuem o maior número de *Stars* (Estrelas) adicionados, indicando o nível de participações no projeto entre os usuários registrados no GitHub.

Para cada conjunto de métricas o algoritmo BSAS foi executado com cinco configurações do valor da distância de tolerância. O valor da distância de tolerância para a criação dos grupos como descrito na Seção 2.3 é muito importante pois tem um efeito direto sobre o número de grupos formados. Se esse valor é muito pequeno grupos desnecessários são criados e se for muito grande é criado um número menor de grupos necessários (KAINULAINEN; JUKKA, 2002).

4.5.1 AGRUPAMENTO COM MÉTRICAS DE *PULL REQUEST*

A Tabela 1 apresenta os resultados obtidos pela execução do algoritmo BSAS com um conjunto de métricas de *pull request*. Na primeira coluna é apresentado os valores utilizados como parâmetro da distância de tolerância. Esses valores foram variados para proporcionar uma formação da quantidade de grupos diferentes em cada execução. Dessa forma, é possível avaliar como os agrupamentos são gerados quando as métricas de *pull request* são utilizadas

[1] http://xmodulo.com/popular-open-source-projects-hosted-github.html

e os valores da distância de tolerância são modificados. Nesse exemplo, os valores da distância de tolerância são sempre incrementados para proporcionar a separação dos projetos em mais grupos, fazendo com que projetos que tenham mais características similares possam ficar agrupados no mesmo grupo.

Na terceira coluna é apresentado a menor quantidade de projetos em um grupo em cada execução. Na quarta coluna o valor apresentado corresponde a maior quantidade de projetos e nas duas últimas é a mediana e a média da quantidade de projetos por grupo.

Distância de tolerância	**Quantidade de grupos**	**Menor quantidade de projetos**	**Maior quantidade de projetos**	**Mediana da quantidade de projetos**	**Média da quantidade de projetos**
90000	22	1	21	1	2,38
150000	16	1	21	1	3
200000	13	1	22	1	3,9
420000	9	1	22	1	5,66
500000	8	1	36	1	6,37

Tabela 1: Resultado do agrupamento com métricas de *pull request*

A Tabela 1 apresentou uma grande variação na quantidade de grupos formados quando é alterado o valor da distância de tolerância. Quando o valor estava em 90000 formou 22 grupos, e em 500000 formou 8.

A fim de ilustrar o resultado obtido, considerando o valor da distância de tolerância igual a 500000, 8 grupos foram formados. A Figura 12 apresenta os grupos, a quantidade de projetos e o valor da coesão de cada um. A coesão é obtida pela soma total da multiplicação das distâncias dos projetos do mesmo grupo. Grupos com coesão 0 possuem um único projeto. Quanto menor o valor da coesão, mais próximos estão esses projetos no grupo formado, logo eles tendem a ser mais similares considerando as métricas de agrupamento usadas (KAINULAINEN; JUKKA, 2002).

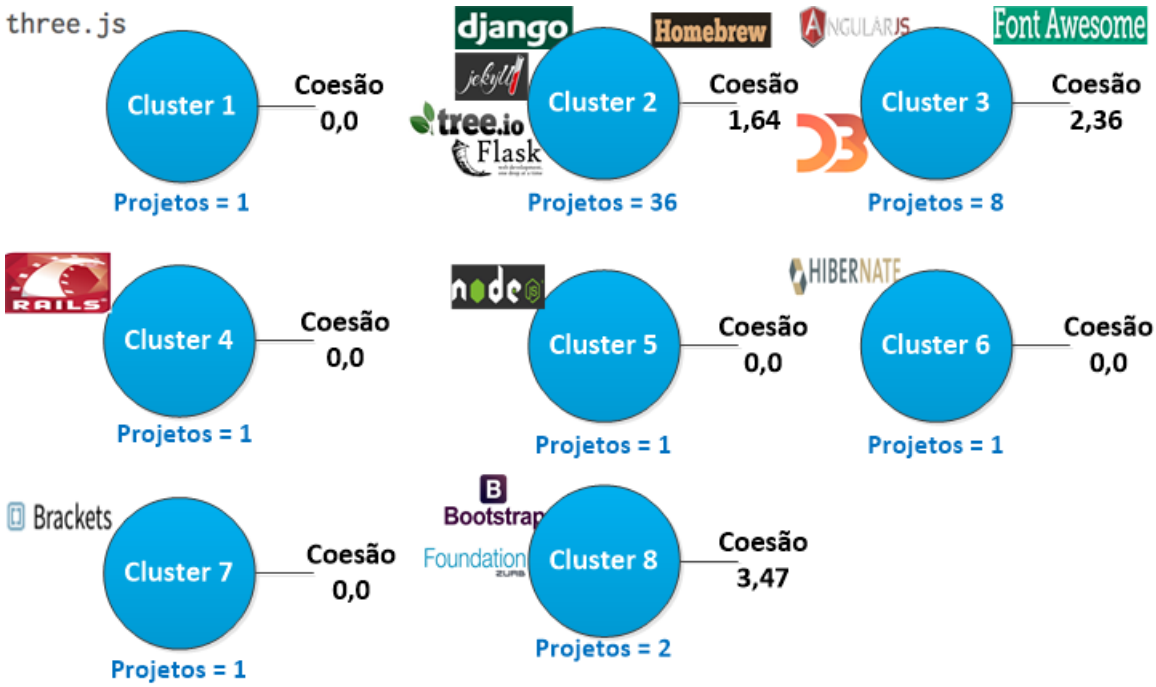

Figura 12: Grupos formados com métricas de *pull request*

Os projetos Bootstrap e Foundation estão agrupados no grupo 8, eles estão entre os maiores projetos da amostra selecionada. Baseado nos valores das métricas de *pull request*, eles são similares na quantidade de linhas adicionadas, alteradas e excluídas. Suponha-se que ao realizar um estudo sobre as métricas de *pull request* é necessário selecionar apenas um dos dois projetos, pois eles são similares nesse contexto, mas para outro conjunto de métricas eles podem não ser similares.

A Tabela 2 apresenta o valor total das métricas de *pull request* obtidas pelo cálculo em 50 projetos do GitHub. A primeira linha apresenta a métrica de teste de inclusão. O projeto Ruby on Rails é o que possui a quantidade máxima de testes, no caso 22293. O projeto HyperTemplate possui a quantidade mínima, 1. Na segunda, terceira e quarta linha da tabela é apresentado os valores das Linhas adicionadas, alteradas e excluídas. Nas duas últimas linhas os valores demostrados são das contribuições amplamente discutidas e a quantidade de comentários em *issues*. Esses valores foram obtidos considerando todos os *pull requests* do projeto.

Métrica	**Quantidade máxima**	**Quantidade mínima**	**Quantidade mediana**	**Quantidade média**
Teste de inclusão	22293	1	79	1004,19
Linhas adicionadas	5768126	317	221555	6249701
Linhas alteradas	10065713	393	345937	3672073
Linhas excluídas	4297587	76	110365	3921152
Contribuições amplamente discutidas	310718	199	1114450	4495717
Comentários em *issues*	40508	3	259	3511,52

Tabela 2: Resultado das métricas de *pull request*

4.5.2 AGRUPAMENTO COM MÉTRICAS DO PERFIL DO USUÁRIO

A Tabela 3 apresenta os resultados obtidos da execução do algoritmo BSAS com um conjunto de métricas do perfil do usuário. A primeira coluna apresenta os valores utilizados para configurar os parâmetros da distância de tolerância. Os valores apresentados podem servir de parâmetro para uma nova execução do algoritmo, e formar uma quantidade maior ou menor de grupos. A quantidade de grupos formados é apresentado na segunda coluna da tabela. Na terceira coluna é apresentado a menor quantidade de projetos em um grupo e na quarta coluna a maior quantidade. Nas duas últimas colunas é apresentada a mediana e a média da quantidade de projetos.

Distância de tolerância	**Quantidade de grupos**	**Menor quantidade de projetos**	**Maior quantidade de projetos**	**Mediana da quantidade de projetos**	**Média da quantidade de projetos**
15000	13	1	24	2	3,92
30000	8	1	28	2,50	6,37
50000	7	1	28	4	7,28
70000	5	1	32	5	10,20
100000	4	1	32	9	12,75

Tabela 3: Resultado do agrupamento com métricas do perfil do usuário

Os valores calculados das métricas do perfil do usuário são menores de que a do *pull request*. Com isso o valor da distância de tolerância para esse conjunto é menor, e a variação da quantidade de grupos formados também. É possível afirmar que aconteceu um agrupamento melhor para esse conjunto de métricas, pois os resultados foram parecidos para cada execução do algoritmo BSAS. Por exemplo, a quantidade maior de projetos em um grupo é próxima para cada caso, e os projetos agrupados são quase sempre os mesmos.

A Figura 13 apresenta o resultado obtido considerando o valor da distância de tolerância igual a 100000, no qual formou 4 grupos. No grupo 1 a sua coesão é 0, pois possui apenas 1 projeto. No grupo 2 foram agrupados 10 projetos, com coesão de 7,80. No grupo 3 foram 7 projetos agrupados e no grupo 4 foram 32 projetos. A coesão do grupo 3 é de 3,80 e do grupo 4 é 3,34. Mesmo o grupo 4 tendo muito mais projetos, o valor da coesão foi menor que o grupo 3 que contêm menos projetos. Isso indica que os projetos do grupo 3 têm mais diferenças considerando as métricas de distância que os projetos agrupados no grupo 4.

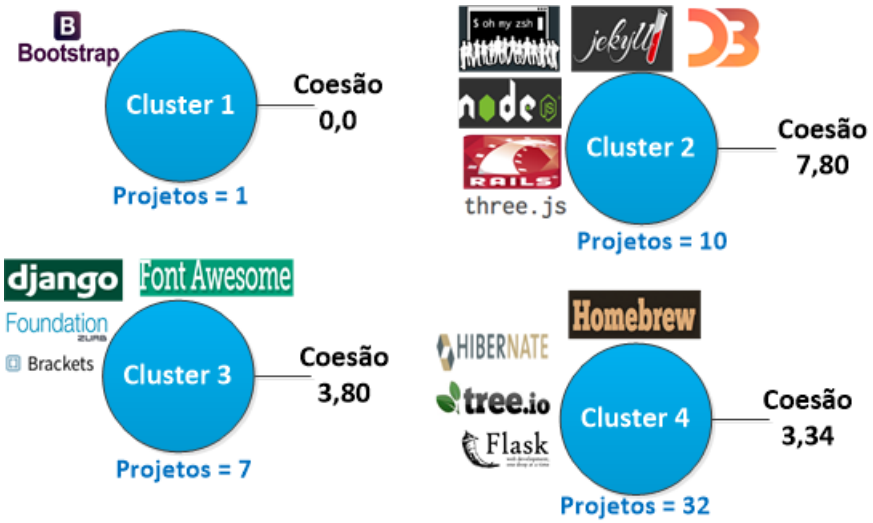

Figura 13: Grupos formados com métricas do perfil do usuário

A Tabela 4 apresenta os valores das métricas do perfil do usuário. As colunas da tabela apresentam os nomes das métricas, os valores de quantidade máxima, mínima, mediana e média que cada métrica possui. A primeira linha apresenta a métrica do número de seguidores que os usuários do projeto possui, a maior quantidade é 509132 do projeto BootStrap e a menor é do projeto HyperTemplate com 22. A segunda linha apresenta a quantidade dos colaboradores ATIVO de um projeto, e a terceira os INATIVOS.

Métrica	**Quantidade máxima**	**Quantidade mínima**	**Quantidade mediana**	**Quantidade média**
Número de seguidores que os usuários possui	509132	22	23947	134034
Colaboradores ATIVO	304	11	17,50	56,25
Colaboradores INATIVO	350	5	21	70,87

Tabela 4: Resultado das métricas do perfil do usuário

Com os resultados obtidos das métricas do perfil do usuário é possível medir informações sociais dos projetos, e chegar a conclusão que por exemplo os projetos Hibernate, Ho-

mebrew, Tree e Flask são similares nesse contexto, pois estão agrupados e possuem o valor da coesão menor que as dos demais. Se utilizar qualquer um desses projetos em um estudo estará abrangendo os quatro, devido a similaridade entre eles.

4.5.3 AGRUPAMENTO COM MÉTRICAS DO *LOG* DO *COMMIT*

A Tabela 5 apresenta os resultados obtidos pela execução do algoritmo BSAS com um conjunto de métricas do *log* do *commit*. Na primeira coluna são apresentados os valores da distância de tolerância. A quantidade de grupos formados é apresentado na segunda coluna. Quanto maior o valor da distância de tolerância menos grupos são formados. O menor valor da distância de tolerância na tabela abaixo é 90000, e a quantidade de grupos formados é 20. O maior valor apresentado é 450000, e formou 8 grupos. Mesmo aumentando o valor da distância a quantidade de grupos permanece em 8.

Distância de tolerância	**Quantidade de grupos**	**Menor quantidade de projetos**	**Maior quantidade de projetos**	**Mediana da quantidade de projetos**	**Média da quantidade de projetos**
90000	20	1	13	1	2,55
120000	16	1	13	1	3
150000	15	1	13	1	3,40
250000	11	1	21	1	4,63
450000	8	1	35	1	6,37

Tabela 5: Resultado do agrupamento com métricas do *log* do *commit*

Os resultados da Tabela 5 variam bastante devido a grande diferença de valores no cálculo das métricas do *log* do *commit*. Nesse contexto existem muitos projetos com valores altos, e outros com valores menores. Entre os agrupamentos formados destaca-se os projetos Angular, Font Awesome e D3 que podem ser considerados similares nesse conjunto de métricas. Os projeto Brackets e Three não foram agrupados com nenhum outro projeto, com isso eles podem ser considerados diferentes dos demais projetos.

A fim de ilustrar o resultado obtido, considerando o valor da distância de tolerância igual a 450000, 8 grupos foram formados. A Figura 14 apresenta os grupos, a quantidade de projetos e o valor da coesão de cada grupo. Os grupos 2 e 3 agrupam a maior quantidade de projetos. O valor da coesão dos dois são 1,60 e 1,37. Os 8 projetos classificados no grupo 3 estão próximos devido ao baixo valor da coesão.

Figura 14: Grupos formados com métricas do log do *commit*

A Tabela 6 apresenta o valor total das métricas do *log* do *commit* obtidas. A primeira linha apresenta os números de *commit*. O projeto com o maior valor é o Ruby on Rails com 64947 e o menor com 100 é o fx2048. O valor da mediana para o números de *commit* é 1939, e a média 5138,58. Na segunda linha o projeto Three.js possui a maior quantidade de linhas alteradas com 10065713. O projeto Docker-Library possui a menor com 393. Linhas alteradas possui mediana igual a 345937, e média 1017085.

Métrica	**Quantidade máxima**	**Quantidade mínima**	**Quantidade mediana**	**Quantidade média**
Número de *commit*	64947	100	1939	5138,58
Linhas alteradas	10065713	393	345937	1017085

Tabela 6: Resultado das métricas do log do *commit*

Entre os resultados obtidos, os projetos Bootstrap e Foundation aparecem novamente agrupados em outro conjunto de métricas. Nesse caso eles são similares nas quantidades de *commits*. Ao realizar estudos com métricas do log do *commit* é possível selecionar apenas um dos dois projetos, pois ambos são similares e será obtido os mesmos resultados.

4.5.4 AGRUPAMENTO COM MÉTRICAS DO PROJETO

É apresentado na Tabela 7 os resultados obtidos pela execução do algoritmo BSAS com um conjunto de métricas do projeto. Na primeira execução do algoritmo o valor do parâmetro da distância de tolerância é igual a 3000. 14 grupos foram gerados, no qual o maior grupo possui 28 projetos. O valor do parâmetro da distância de tolerância foi aumentado para visualizar outros agrupamentos. Em 8000 foram gerados 6 grupos. Em 13000 gerou 4 grupos, no qual o maior grupo possui 30 projetos. Nesse estudo, mesmo o valor do parâmetro da distância de tolerância sendo superior a 13000, a quantidade de grupos formados permanece em 4.

Distância de tolerância	**Quantidade de grupos**	**Menor quantidade de projetos**	**Maior quantidade de projetos**	**Mediana da quantidade de projetos**	**Média da quantidade de projetos**
3000	14	1	28	1	3,64
5000	10	1	29	1	5,10
8000	6	1	31	1,50	8,50
10000	4	3	31	9,50	12,75
13000	4	3	30	9,50	12,75

Tabela 7: Resultado do agrupamento com métricas do projeto

Os resultados apresentados na Tabela 7 mostram uma pequena diferença na formação de grupos para o conjunto de métricas do projeto. Para esse contexto a maioria dos projetos estão próximos, são mais similares. É possível observar um melhor agrupamento e uma boa separação dos projetos por grupo.

Para ilustrar os grupos formados com o valor da distância de tolerância igual a 13000 é apresentada a Figura 15. Quatro grupos foram formados, sendo que o primeiro agrupou 3 projetos, o segundo agrupou 16, o terceiro 30 e o último apenas 1 projeto. Os projetos Rails, Bootstrap e Angular.js pertencem ao grupo 1, e são próximos na quantidade de *stars* (estrelas) e *issues* fechada. O valor da coesão do grupo 1 é 3,57. No segundo grupo a coesão é 2,76 e no terceiro é 1,09.

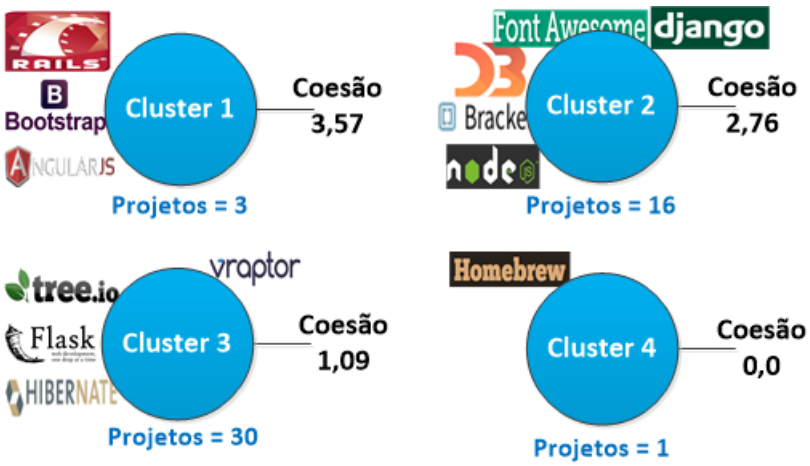

Figura 15: Grupos formados com métricas do projeto

A Tabela 8 apresenta os valores obtidos pelas métricas do projeto. A primeira métrica representada é a de *Stars* (Estrelas). O projeto com a maior quantidade de estrelas é o Bootstrap com 32264, ele pertence ao grupo 1. HyperTemplate pertence ao grupo 3, é o projeto que possui a menor quantidade de estrelas, 16. A segunda métrica representada é a idade do repositório. O projeto Ruby on Rails pertence ao grupo 1, possui 86 meses e é o mais antigo da amostra coletada. Com a idade de 7 meses, Robusta-code/swing é o mais novo e pertence ao grupo 3.

O projeto Font-Awesome pertence ao grupo 2 e é o projeto com a maior quantidade de *issues* em aberto, 2733. Com a menor quantidade o Caelum/guj.com.br possui 2 e pertence ao grupo 3. A maior quantidade de *issues* fechada é do projeto Homebrew, e a menor do HyperTemplate. Homebrew possui 38710 e pertence ao grupo 4. HyperTemplate possui 4, e pertence ao grupo 3.

Métrica	**Quantidade máxima**	**Quantidade mínima**	**Quantidade mediana**	**Quantidade média**
Stars (Estrelas)	32264	16	1387	8021,25
Idade do repositório	86	7	43	43,78
Número de *issues* em aberto	2733	2	59	243,52
Número de *issues* fechada	38710	4	287	2731,80

Tabela 8: Resultado das métricas do projeto

O projeto Homebrew é o único projeto que não está agrupado com nenhum outro nos resultados obtidos das métricas do projeto, suponha-se que ele não é similar a nenhum outro nesse contexto. Os projetos Rails, Bootstrap e Angular estão agrupados, porém não possuem uma boa similaridade devido ao valor da coesão ser alto. Já os projetos Vraptor, Tree, Flask

e Hibernate estão muito próximo no valor da coesão, e podem ser considerados similares para esse contexto.

4.5.5 AGRUPAMENTO COM TODAS MÉTRICAS

A Tabela 9 apresenta os resultados obtidos na execução do algoritmo BSAS com o cálculo de todas métricas. Com o valor da distância de tolerância igual a 700000, é formado 8 grupos. Mesmo aumentando o valor da distância, a quantidade de grupos permanece em 8. A menor quantidade de projetos em um único grupo é 1, e a maior é 35. O menor valor da distância de tolerância utilizado é de 50000, gerou 30 grupos e a maior quantidade de projetos em um único grupo é 10. Com o valor da distância de tolerância em 300000 são formados 10 grupos, a menor quantidade de projetos em um único grupo é 1 e a maior 22.

Distância de tolerância	**Quantidade de grupos**	**Menor quantidade de projetos**	**Maior quantidade de projetos**	**Mediana da quantidade de projetos**	**Média da quantidade de projetos**
50000	30	1	10	1	1,70
100000	20	1	11	1	2,42
300000	10	1	22	1	4,72
500000	8	1	35	1	6,37
700000	8	1	35	1	6,37

Tabela 9: Resultado do agrupamento com todas métricas

A Figura 16 apresenta os 8 grupos formados com o valor da distância de tolerância igual a 700000. O grupo 2 é o que agrupou a maior quantidade de projetos, 35. O grupo 3 agrupou 2 projetos, e o grupo 4 agrupou 8. Os grupos 1, 5, 6, 7 e 8 agruparam apenas 1 projeto. A menor coesão entre os grupos que possui mais de um projeto é do grupo 2 com o valor de 1,59.

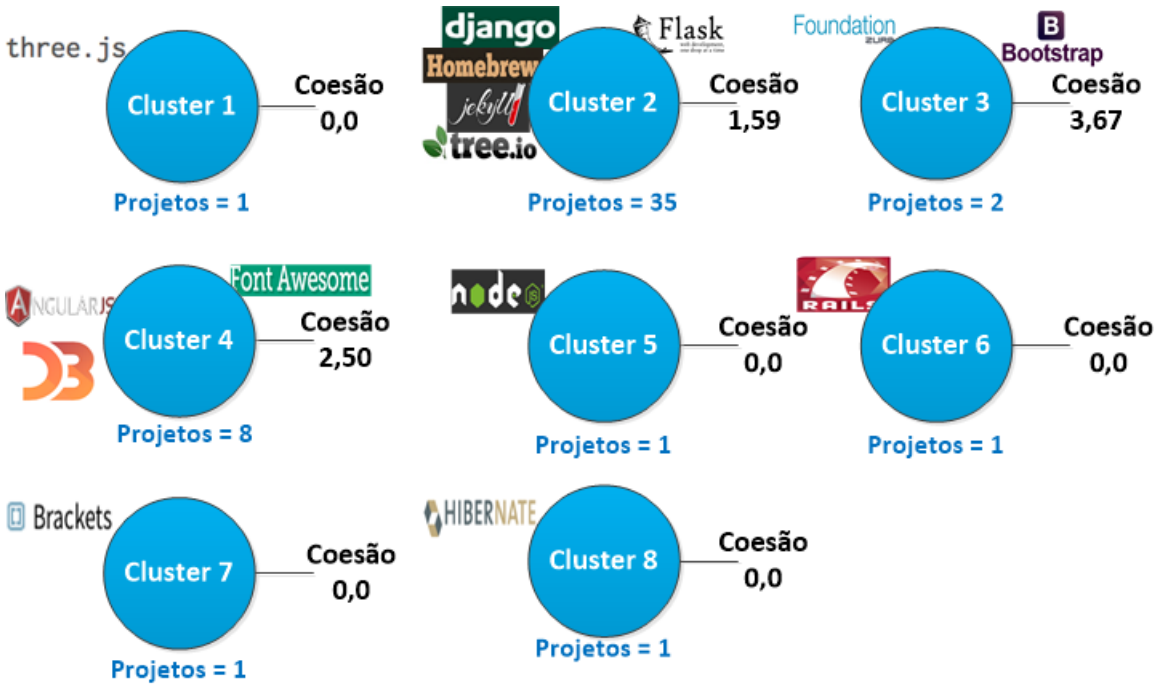

Figura 16: Grupos formados com todas métricas

Os resultados dos agrupamentos apresentam que quando os valores dos vetores de métricas são maiores, o valor do parâmetro da distância de tolerância e a quantidade de grupos formados tendem a ser maiores. Um exemplo é os resultados do agrupamento com métricas do *log* do *commit* que gerou 8 grupos e o valor da distância de tolerância chegou em 450000. Quando os valores da quantidade de métricas calculadas e dos resultados são menores, a quantidade de grupos formados é menor e o valor da distancia de tolerância também. Um exemplo é os resultados das métricas do perfil do usuário.

Os projetos Bootstrap e Zurb/Foundation foram agrupados nos resultados obtidos pelas métricas de *pull request*, *log* do *commit* e de todas métricas. Nos agrupamentos dos resultados das métricas de *pull request* eles foram agrupados no grupo 8 e nos resultados das métricas do log do *commit* foram agrupados no grupo 7. Nos agrupamentos com todas métricas eles foram agrupados no grupo 3. No primeiro agrupamento, grupo 8, a coesão é de 3,47. No segundo agrupamento a coesão é de 2,39 e no terceiro é 3,67. Entre os três agrupamento criados, possui maior similaridade em valores de métricas do *log* do *commit*, que a coesão é de 2,39. Entre os 50 projetos da amostra selecionada, Bootstrap e Zurb/Foundation estão entre os maiores. Esses projetos são próximos nos valores das métricas do número de *issues* em aberto, idade do repositório e também nas quantidades de linhas adicionadas, alteradas e excluídas. A Tabela 10 abaixo apresenta os valores das métricas citadas para os dois projetos.

Projeto	***Issues* em aberto**	**Idade do repositório**	**Linhas adicionadas**	**Linhas alteradas**	**Linhas excluídas**
Bootstrap	121	46	1191817	2128463	936646
Foundation	102	44	1112373	2093031	980658

Tabela 10: Resultados de métricas entre os projetos Bootstrap e Foundation

Na comparação de grupos formados com métricas diferentes, por exemplo o grupo 2 das métricas de *pull request* agrupou 36 projetos com o valor da coesão em 1,64. O mesmo grupo 2 das métricas do *log* do *commit* agrupou 35 projetos com o valor da coesão em 1,60. Mesmo realizando a comparação com métricas de categorias diferente os resultados em valor de coesão e quantidade de projetos foi muito parecido.

Outro exemplo é o projeto Hibernate que ficou sozinho em três conjuntos de métricas. Nas métricas de *pull request* ele ficou no grupo 6. Nas métricas do *log* do *commit* no grupo 5, e no resultado de todas métricas no grupo 8.

A Tabela 11 apresenta a quantidade de vezes que um projeto ficou agrupado com outro. Com base nos resultados obtidos da execução do algoritmo BSAS para todos grupos de métricas, é selecionado para essa amostra apenas alguns projetos entre os 50 comparados. Os projetos Brackets e Django ficaram duas vezes no mesmo grupo. Brackets e D3 ficaram apenas uma vez. Brackets e Font Awesome duas. O projeto Django foi agrupado três vezes com os projetos Tree, Flask e Homebrew. Apenas uma vez com o projeto D3 e duas com o Font Awesome. O projeto Tree foi agrupado quatro vezes com o projeto Flask e Homebrew. O projeto Flask foi agrupado quatro vezes com Homebrew. O projeto D3 é agrupado quatro vezes com o projeto Font Awesome.

	Brackets	Django	Tree	Flask	Homebrew	D3	Font Awesome
Brackets		2				1	2
Django	2		3	3	3	1	2
Tree		3		4	4		
Flask		3	4		4		
Homebrew		3	4	4			
D3	1	1					4
Font Awesome	2	2				4	

Tabela 11: Quantidade de agrupamento entre projetos

Conforme apresentado na Tabela 11, é possível concluir que quanto maior a quantidade

de vezes que um projeto ficou agrupado com outro, mais similares eles são. Com os resultados das métricas, e o agrupamento dos projetos é possível os pesquisadores analisarem e escolher quais projetos desejam incluir no estudo.

5 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou uma ferramenta para possibilitar a coleta de projetos do GitHub, calcular métricas de *pull request*, do perfil do usuário, do *log* do *commit* e do projeto para realizar o agrupamento de projetos similares utilizando o algoritmo BSAS.

A contribuição deste trabalho tem valor na área de Mineração de Repositórios de Software e auxiliar os pesquisadores a escolher projetos similares para seus estudos, permitindo indicar uma possível generalização dos seus resultados para outros contextos. Com isso, é possível que pesquisadores selecionem apenas alguns projetos de uma amostra, no qual garante que esses projetos são semelhantes aos outros, e se ele utilizar apenas os projetos selecionados no seu estudo ele esta cobrindo toda aquela amostra, permitindo uma boa análise com uma quantidade menor de projetos, economizando tempo e esforço para realizar o estudo empírico.

Por fim é apresentado um estudo exploratório do agrupamento de projetos com os dados obtidos por cada conjunto de métrica. No estudo exploratório é demonstrado diversas maneiras e parâmetros de como executar o algoritmo BSAS para realizar o agrupamento dos projetos, e obter resultados com uma quantidade maior ou menor de grupos formados.

Como trabalhos futuros, pretende-se implementar novas métricas para possibilitar novas comparações, além de colocar uma correlação entre elas. Realizar alteração no módulo de coleta da ferramenta para não precisar coletar projetos manualmente, adicionando compatibilidade com a base de dados disponibilizada pelo GHTorrent[1] com informações dos projetos do GitHub. Realizar comparação dos resultados obtidos com o trabalho de Nagappan et al. (2013), como citado na Seção 2.3. Implementar e comparar outros algoritmos de agrupamento.

[1] http://ghtorrent.org/

REFERÊNCIAS

DABBISH, L.; STUART, C.; TSAY, J.; HERBSLEB, J. Social coding in github: Transparency and collaboration in an open software repository. In: **Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work**. New York, NY, USA: ACM, 2012. (CSCW '12), p. 1277–1286. ISBN 978-1-4503-1086-4. Disponível em: `http://doi.acm.org/10.1145/2145204.2145396`.

GITHUB: Social Coding: Site. 2015. Disponível em: `http:///github.com/`. Acesso em: 22 Jun. 2015.

HASSAN, A. E.; XIE, T. Mining software engineering data. In: **Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2, ICSE 2010, Cape Town, South Africa, 1-8 May 2010**. [s.n.], 2010. p. 503–504. Disponível em: `http://doi.acm.org/10.1145/1810295.1810451`.

JUNIOR, D. N. R. Uma ferramenta para mineração de dados de projetos de software livre e criação de redes sócio-técnicas. 2013.

JURECZKO, M.; MADEYSKI, L. Towards identifying software project clusters with regard to defect prediction. In: **Proceedings of the 6th International Conference on Predictive Models in Software Engineering**. New York, NY, USA: ACM, 2010. (PROMISE '10), p. 9:1–9:10. ISBN 978-1-4503-0404-7. Disponível em: `http://doi.acm.org/10.1145/1868328.1868342`.

KAINULAINEN, J.; JUKKA, J. **Clustering Algorithms: Basics and Visualization**. 2002.

KHADKE, N.; TEH, M. H.; SHEN, M. Predicting acceptance of github pull requests. 2012.

KOSCIANSKI, A.; SOARES, M. dos S. (Ed.). **Qualidade de Software - 2ª Edição**. [S.l.]: Novatec, 2007. ISBN 342-3-540-76349-1.

MARLOW, J.; DABBISH, L.; HERBSLEB, J. D. Impression formation in online peer production: activity traces and personal profiles in hub. In: BRUCKMAN, A.; COUNTS, S.; LAMPE, C.; TERVEEN, L. G. (Ed.). **CSCW**. [S.l.]: ACM, 2013. p. 117–128. ISBN 978-1-4503-1331-5.

NAGAPPAN, M.; ZIMMERMANN, T.; BIRD, C. Diversity in software engineering research. In: MEYER, B.; BARESI, L.; MEZINI, M. (Ed.). **ESEC/SIGSOFT FSE**. [S.l.]: ACM, 2013. p. 466–476. ISBN 978-1-4503-2237-9.

OHLOH: the open source network: Site. 2015. Disponível em: `http:///ohloh.net/`. Acesso em: 19 Jun. 2015.

SEGARAN, T. **Programming Collective Intelligence**. First. [S.l.]: O'Reilly, 2007. ISBN 9780596529321.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. **Introdução ao data mining - Mineração de dados**. 1ª edição. ed. [S.l.]: Ciência moderna, 2009.

TSAY, J.; DABBISH, L.; HERBSLEB, J. D. Influence of social and technical factors for evaluating contribution in github. In: JALOTE, P.; BRIAND, L. C.; HOEK, A. van der (Ed.). **ICSE**. [S.l.]: ACM, 2014. p. 356–366. ISBN 978-1-4503-2756-5.

ZAÏANE, O. R.; FOSS, A.; LEE, C.; WANG, W. On data clustering analysis: Scalability, constraints, and validation. In: **Advances in Knowledge Discovery and Data Mining, 6th Pacific-Asia Conference, PAKDD 2002, Taipei, Taiwan, May 6-8, 2002, Proceedings**. [s.n.], 2002. p. 28–39. Disponível em: `http://dx.doi.org/10.1007/3-540-47887-6_4`.

ZIMMERMANN, T.; NAGAPPAN, N.; GALL, H.; GIGER, E.; MURPHY, B. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: VLIET, H. van; ISSARNY, V. (Ed.). **ESEC/SIGSOFT FSE**. [S.l.]: ACM, 2009. p. 91–100. ISBN 978-1-60558-001-2.