PUC Minas
A B A K Ó S
Instituto de Ciências Exatas e Informática

Aplicação de Técnicas de Inteligência Analítica em Repositórios de *Software*[*]

Application of Software Analytics Techniques in Software Repositories

Bruno Rafael de Oliveira Rodrigues[1]
Fernando Silva Parreiras[2]

Resumo

A inteligência analítica na Engenharia de *Software* permite analisar os dados contidos nos diversos repositórios com a finalidade de apresentar decisões fundamentadas aos engenheiros de *software*. Com a intenção de promover a utilização da inteligência analítica na Engenharia de *Software*, este artigo responde a questão: é possível utilizar a inteligência analítica para descobrir quais arquivos de código fonte do sistema têm sido alterados e quais os motivos de suas alterações? Por meio das técnicas de modelo de tópicos e regra de associação foi realizado um estudo com o repositório de código fonte de um sistema de *software* livre, o Jenkins e desenvolvido um protótipo de um sistema a partir dessas técnicas. Com o auxílio de um grupo focal formado por profissionais da área de desenvolvimento de sistemas foi possível avaliar o protótipo desenvolvido. Apura-se que o método apresentado, neste trabalho, permite identificar quais arquivos do sistema estão sendo alterados e a justificativa para essa modificação, facilitando o mapeamento dos arquivos do sistema por assunto, o planejamento de refatorações, *builds* e o entendimento da estabilidade e evolução do *software*.

Palavras-chave: Inteligência Analítica. Engenharia de *Software*. Mineração de Repositórios de *Software*.

[*]Submetido em 03/10/2017 - Aceito em 05/03/2018

[1]Mestre em Sistemas de Informação e Gestão do Conhecimento pelo Programa de Pós-Graduação em Sistemas de Informação e Gestão do Conhecimento da Universidade FUMEC, – brunorodriguesti@yahoo.com.br

[2]Doutor em Ciência da Computação pela Universität Koblenz Landau, professor e coordenador do Programa de Pós-Graduação em Sistemas de Informação e Gestão do Conhecimento da Universidade FUMEC, – fernando.parreiras@fumec.com.br

Abstract

Software analytics enables data analysis in the various software repositories in order to provide well-founded decisions for software engineers. With the intention of promoting the use of software analytics this paper answers the question: Can software analytics find out which system source files have been altered and what are the reasons for its alterations? Through techniques of topics model and the association rule, a study was carried out over source code repository of the open source software, the Jenkins, and we developed a prototype system from these techniques. With the help of a focus group formed by professionals in software development it was possible to evaluate the prototype developed. We verified that the method presented in this study allows to identify which files of the system are being altered and why, facilitating the mapping of the files of the software by subject, the planning of refactoring, *builds* and software stability and evolution.

Keywords: Software Analytics. Software Engineering. Mining Software Repositories.

1 INTRODUÇÃO

A inteligência analítica é descrita como uma técnica que faz uso de análises, dados e raciocínio sistemático para tomar decisões (BUSE; ZIMMERMANN, 2012). Demonstrou ser uma técnica valiosa para a análise de processos de negócios em vários domínios. Na Engenharia de *Software*, seu uso tem crescido de modo substancial na identificação de padrões e apoio à tomada de decisão (ZHANG et al., 2011). Contudo, é comum que decisões tomadas no desenvolvimento de *software* sejam baseadas somente na intuição do engenheiro de *software*, sem utilização de ferramentas ou indicadores para analisar o projeto (HASSAN, 2008).

Para embasar decisões em projetos de *software*, é necessário entender o que aconteceu com o projeto durante seu desenvolvimento ou manutenção. Dessa maneira, este trabalho tem o objetivo de responder a seguinte questão: é possível utilizar a inteligência analítica para descobrir quais arquivos de código fonte do sistema têm sido alterados e quais os motivos de suas alterações? Para responder a essa questão, o presente trabalho propõe identificar os tópicos mais relevantes nos comentários dos *commits*, feitos pelos desenvolvedores e relacioná-los com os arquivos mais alterados no repositório de código fonte de acordo com cada tópico. São utilizadas as técnicas de modelos de tópicos, mais especificamente o *Latent Dirichlet Allocation* (LDA) para identificar os tópicos mais relevantes nas mensagens dos *commits* e a regra de associação *Apriori* para se descobrir os arquivos relacionados com cada assunto.

O uso de modelos de tópicos e regra de associação na Engenharia de *Software* são comumente abordados na literatura. Modelos tópicos, em geral, são aplicados para sumarizar e identificar assuntos em conjuntos de textos (ALALI et al., 2008) (HU et al., 2015) e as regras de associação buscam por padrões de relacionamento em um conjunto de dados (DIT et al., 2014) (ZIMMERMANN et al., 2005). Contudo, a união dessas técnicas aplicadas ao histórico de versão torna possível associar os assuntos, com os quais os desenvolvedores estiverem trabalhando, aos seus respectivos arquivos. Com isso, facilita-se a rastreabilidade no código fonte, auxiliando decisões na análise de impacto, e aos desenvolvedores e gestores compreenderem melhor o sistema sem a necessidade de explorar o código fonte.

Este trabalho aplica tais técnicas sobre o repositório de código fonte do sistema de *software* livre Jenkins. Por meio dessa análise, foi gerado um protótipo de um sistema que permite analisar os dados do repositório de código fonte do projeto. Para avaliar o método proposto neste trabalho, foi realizado um grupo focal com desenvolvedores e gestores de projeto de uma empresa de grande porte no ramo de tecnologia da informação.

Como contribuições deste trabalho podem ser destacadas:

- A proposta de uma metodologia capaz de sumarizar os arquivos alterados no código fonte de um sistema e os principais motivos de suas alterações.
- A criação de um modelo de um protótipo de um sistema de inteligência analítica que auxilie os engenheiros de *software* a visualizar a evolução do *software* e as contribuições dos desenvolvedores. Tal protótipo fornece informações úteis para planejar *builds*, refato-

rações, gerenciar equipes, avaliar estabilidade e maturidade de *software*, além de agregar conhecimento do projeto à equipe.

- Como contribuição secundária, fonte extraídos do sistema Jenkins utilizados neste trabalho.

Este artigo está estruturado da seguinte maneira: a Seção 2 a apresenta os conceitos necessários para o entendimento deste trabalho; a Seção 3, a abordagem; a Seção 4, a aplicação do método em uma prova de conceito; a Seção 5, o grupo focal; a Seção 6, os resultados do grupo focal; a Seção 7, as limitações do trabalho e a Seção 8, a conclusão e a proposta de trabalhos futuros.

2 CONCEITUAÇÃO

2.1 Inteligência Analítica

A inteligência analítica permite aos engenheiros de *software* realizarem a exploração e a análise de dados do sistema, a fim de se obter informações detalhadas e baseadas em dados do próprio *software*, auxiliando nas decisões e condução do projeto e seus serviços (ZHANG et al., 2011). Pesquisas têm relatado o uso da inteligência analítica, por exemplo, para entender sistemas de *software*, propagação de mudanças, predição e identificação de defeitos, dinâmica de equipes, experiência do usuário, reusabilidade e automação de técnicas de mineração de repositórios (HASSAN, 2008). Ela permite melhorar a qualidade do *software* (ZHANG et al., 2011), capacitar os indivíduos e equipes entre outras atividades.

Projetos de *software* tendem a continuar crescendo em tamanho e complexidade, tornando maior o esforço para analisar e interpretar os dados disponíveis em repositórios de *software*. Com o uso da inteligência analítica na Engenharia de *Software*, métricas são extraídas e seus dados analisados com a utilização de técnicas estatísticas e computacionais.

A inteligência analítica na Engenharia de *Software* torna possível a utilização de indicadores para a tomada de decisão. Sem esses indicadores, desenvolvedores e gestores contam apenas com a intuição e a experiência para embasar decisões (BUSE; ZIMMERMANN, 2012).

2.2 Modelo de Tópicos

Modelos de tópicos são modelos probabilísticos que procuram, encontrar palavras relevantes em um texto. Um tópico funciona como uma distribuição probabilística sobre as palavras (STEYVERS; GRIFFITHS, 2007). O LDA é um modelo de tópicos probabilístico. A ideia básica é que os documentos são representados de forma aleatória ao longo dos tópicos, e cada tópico é caracterizado por uma distribuição de palavras.

Os modelos de tópicos têm se mostrado bastante úteis em pesquisas de inteligência analítica na Engenharia de *Software* como, por exemplo, para automatizar a estrutura de dados textuais encontrados em repositórios de *software*, analisar a evolução de *software*, descobrir tópicos em sites de discussão de perguntas e respostas técnicas sobre desenvolvimento de *software*, como o StackOverflow (BARUA et al., 2014) entre outros. Um tópico pode ser definido como uma coleção de palavras que ocorrem simultaneamente em um documento (THOMAS, 2011), ou seja, palavras que são relevantes para o entendimento do documento.

2.3 Regras de Associação

As regras de associação auxiliam na descoberta de padrões exibindo os valores que ocorrem juntos em um determinada amostra de dados (AGRAWAL et al., 1994). Esses dados são chamados de itens. Cada item faz parte de uma transação (AGRAWAL et al., 1994). Uma transação pode ser compreendida como um registro no banco de dados. Depois de ler uma transação, o algoritmo determina quais itens foram mais encontrados em todas as transações. O algoritmo *Apriori* foi proposto para descobrir padrões em grandes coleções de itens (AGRAWAL et al., 1994).

O uso de algoritmos na inteligência analítica permite automatizar e descobrir padrões para diversas atividades da Engenharia de *Software* como, por exemplo, para analisar impacto em projetos de *software* (DIT et al., 2014), ou seja, tornar possível verificar os artefatos que tenham dependência entre si.

2.4 Trabalhos Relacionados

Trabalhos envolvendo repositórios de *software* têm sido frequentemente explorados na literatura. Hu et al. (2015) utilizaram a técnica *Dynamic Topic Model* para minerar os comentários dos *commits* com a finalidade de capturar a evolução do *software* em perspectivas úteis aos desenvolvedores e gerentes, mostrando as mudanças de tópicos de desenvolvimento de diferentes aspectos em um intervalo de tempo. Já Hindle et al. (2011) usaram o modelo de tópicos LDA para extrair tópicos dos *commits* e, por meio de técnicas de aprendizado de máquinas, rotularam as atividades de manutenção de *software* como manutenibilidade, funcionalidade, portabilidade, eficiência, usabilidade e confiabilidade.

Além do modelo de tópicos, algoritmos de regras de associação têm sido objeto de estudo para fazer recomendações de alteração no sistema. Zimmermann et al. (2005) exploraram a técnica de regra de associação para criar a ferramenta Rose, que permite identificar as alterações no código fonte fortemente relacionadas. A ferramenta faz recomendações de alterações dos arquivos, evitando erros de mudanças incompletas, além de detectar acoplamento.

Hsu e Lin (2011), também, usaram das regras de associação para desenvolver a ferra-

menta *Mining API Code snippet for code reuse* (MACs), que extrai padrões de utilização das interfaces de programação de aplicações (*Application Programming Interface* - API) que estão fragmentadas no código, auxiliando, principalmente, novatos a trabalharem com essas API's.

Assim como (HU et al., 2015) e Hindle et al. (2011), o presente trabalho, também, utilizou técnicas de mineração de textos nos comentários dos *commits* para mostrar o que está acontecendo no projeto. Contudo, neste trabalho, os assuntos mais relevantes e que são constantemente alterados foram capturados e mapeados, por meio do algoritmo de regras de associação, tal como fizeram Zimmermann et al. (2005) e Hsu e Lin (2011). Diferente deles, a intenção não é recomendar alterações e uso de código fonte, mas sim mapear as alterações de acordo com os tópicos encontrados, que permitam analisar impactos e auxiliar o desenvolvedor a identificar arquivos que devam ser modificados de acordo com o assunto, engajando o engenheiro de *software* no projeto trabalhado.

3 ABORDAGEM

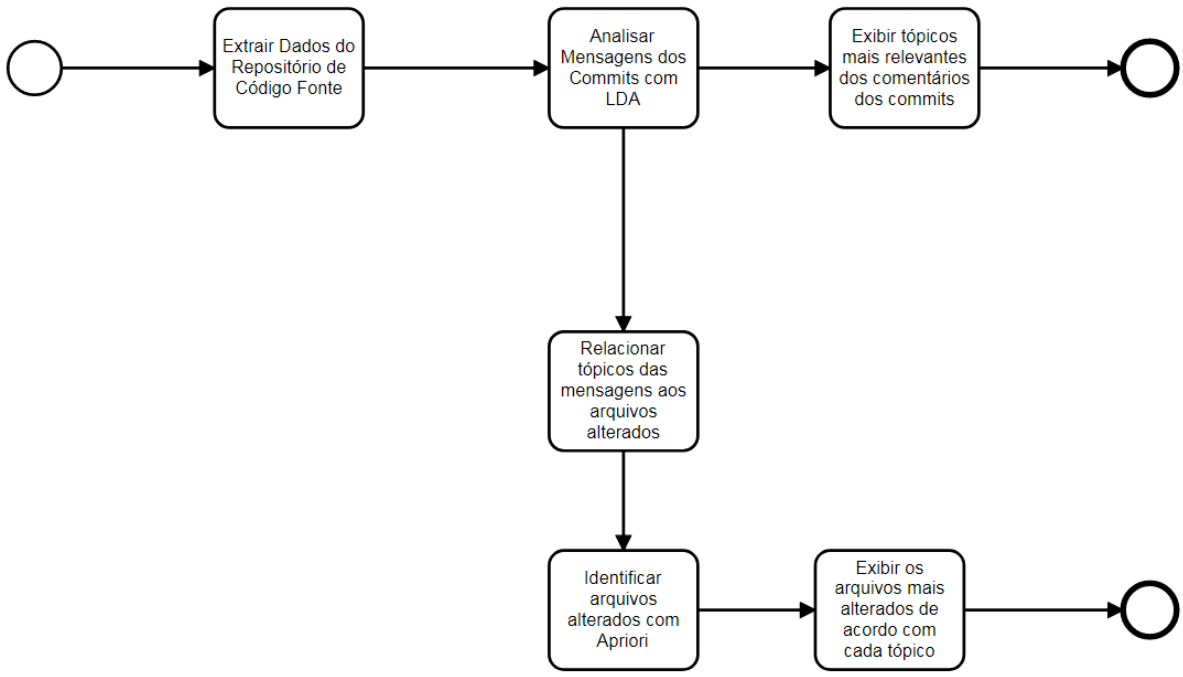
Reconhecendo a preocupação dos engenheiros de *software* em saber o que tem acontecido ao projeto (BUSE; ZIMMERMANN, 2012), este trabalho tem por objetivo expor os arquivos alterados durante o seu desenvolvimento e indicar porque eles foram alterados. Para atender a esse propósito, foi utilizado o modelo de tópicos LDA aplicados aos textos dos comentários feitos durante os *commits* dos desenvolvedores, identificando os tópicos mais relevantes nos comentários. Por meio dos tópicos obtidos nos comentários dos *commits* pelo LDA, pode-se entender o que tem acontecido no projeto ao longo do tempo. Ou seja, é possível ver os assuntos mais comentados pelos próprios desenvolvedores ao realizarem alterações no sistema.

A fim de mapear os arquivos do código fonte de acordo com cada um dos tópicos encontrados, utilizou-se o algoritmo de regra de associação *Apriori*. Dessa maneira, os tópicos são relacionados aos arquivos de código fonte mais alterados em cada tópico. Os tópicos nos apresentam o motivo das alterações nesses arquivos e o *Apriori* auxilia a identificar os arquivos sempre modificados por cada tópico.

Analisando os comentários dos commits por meio do LDA e associando-os por meio do *Apriori*, é possível saber os motivos mais frequentes que levam determinados arquivos a serem comumente alterados e mapeá-los por assunto. Com isso, espera-se facilitar o entendimento do sistema para novos desenvolvedores e gerentes, facilitando a análise de impactos.

Para realizar essas análises, este trabalho seguiu os passos conforme apresentado na Figura 1, onde é possível verificar o processo aplicado nesta pesquisa. Esse processo pode ser resumido em três fases: extração, análise e apresentação dos dados. Na fase de extração são obtidos os dados do repositório de código fonte necessários. Na fase de análise, são aplicadas as técnicas de inteligência analítica, mais especificamente o LDA, que gera os tópicos. O *Apriori* busca o relacionamento entre os dos tópicos encontrados nos arquivos mais alterados com determinado tópico e; por fim, são apresentados os resultados de forma textual.

Figura 1 – Processo para análise de repositório de *software*



Fonte: Elaborada pelos autores.

4 PROVA DE CONCEITO

Para demonstrar a utilização das técnicas de inteligência analítica aplicadas à Engenharia de *Software*, realizou-se um estudo de caso com o repositório de código fonte do sistema Jenkins. O Jenkins é um servidor de integração contínua escrito na linguagem Java (JENKINS, 2011). Esse sistema foi escolhido porque seu conteúdo está inteiramente disponível ao público e por possuir uma comunidade bastante ativa. Essas características fornecem condições ideais para promover os estudos sobre inteligência analítica na Engenharia de *Software* (ROBLES et al., 2014).

O primeiro passo para se realizar a análise de um projeto de *software* é obter os dados de seu repositório, conforme Figura 1. No presente estudo, foram obtidos os dados do repositório de código fonte da *branch master* do Jenkins, disponível pelo sistema de controle de versão Git (JENKINS, 2011). O repositório foi clonado para se trabalhar de forma local e a extração dos dados do repositório contou com a utilização da ferramenta CvsAnaly2, que permitiu a automatização dessa tarefa. O CVSAnaly2 extrai os dados do repositório de controle de fonte dos repositórios CVS, Subversion e Git, que, depois de extraídas ficam armazenadas em um banco de dados relacional, por padrão, o MySQL (ROBLES et al., 2014). Por meio de ferramentas de extração de dados de repositórios, profissionais e pesquisadores podem realizar diversas análises, sem ter de que dispensar tanto tempo no processo de extração dos dados, podendo elevar seus esforços na análise dos dados.

A base de dados do sistema Jenkins utilizada como objeto de estudo deste trabalho, continha 24.417 comentários de commits entre 05-11-2006 e 11-03-2016. Contava, também,

com 9717 arquivos e o total 586 desenvolvedores que contribuíram no projeto.

Para a análise do modelo de tópicos, foi utilizada a API Java Mallet em sua versão 2.07 (MCCALLUM, 2002). Essa API fornece condições de analisar textos com o modelo de tópicos LDA. Modelos de tópicos como o LDA têm sido utilizados para automatizar estrutura de dados textuais encontrados em repositórios de *software*, descobrindo um conjunto de temas dentro dos documentos (THOMAS, 2011).

A API Mallet foi configurada para 1000 iterações e 40 tópicos. O número de iterações é a quantidade de vezes que o modelo interage com a amostra, proporcionando um equilíbrio entre o tempo de execução e a convergência do modelo. Com o valor configurado, obteve-se boa estabilidade de acordo com os resultados obtidos. Em relação à quantidade de tópicos, um número muito pequeno de tópicos pode ser difícil de ser encontrado quando se tem muitos textos, por outro lado, um número grande pode valorizar tópicos que não são realmente relevantes. Assim, julgou-se por bem definir um valor médio para a quantidade de tópicos. Essa configuração permite obter um resultado médio desejado para a análise dos dados.

Para reduzir o ruído de termos encontrados nos comentários dos *commits*, utilizou-se um arquivo de *stop words* , ou seja, um arquivo que contém palavras comuns da língua inglesa como "*a*", "*the*"e "*is*", as quais não ajudam a gerar tópicos significativos (BARUA et al., 2014). Portanto, essas palavras são ignoradas na análise. Nesse arquivo de *stop words*, foram adicionadas as palavras apresentadas por (ALALI et al., 2008), que analisaram 9 projetos e relacionaram os termos mais utilizados nas mensagens dos *commits*. Em geral, essas palavras são frequentemente encontradas nos comentários dos desenvolvedores, portanto, não agregam valor a análise proposta no presente trabalho, uma vez que essas palavras já são esperadas de serem encontradas nos comentários. A relação de palavras adicionadas ao arquivo de *stop words* pode ser vista no Quadro 1.

Quadro 1 - Palavras mais usadas nos comentários dos commits

fix, add, test, file, new, support, chang, change, bug, patch, code, remov, set, work, update, get, error, build, function, typo, call, message, includ, path, need, merge, use, doc

Fonte: ALALI; KAGDI; MALETIC, 2008.

Com a finalidade de descobrir os arquivos constantemente alterados de acordo com os termos encontrados nos *commits*, foi utilizado o algoritmo de regra de associação: *Apriori* por meio da biblioteca SPMF, que é uma biblioteca de mineração de dados de código aberto escrita em Java, especializada em mineração de padrões (FOURNIER-VIGER et al., 2014).

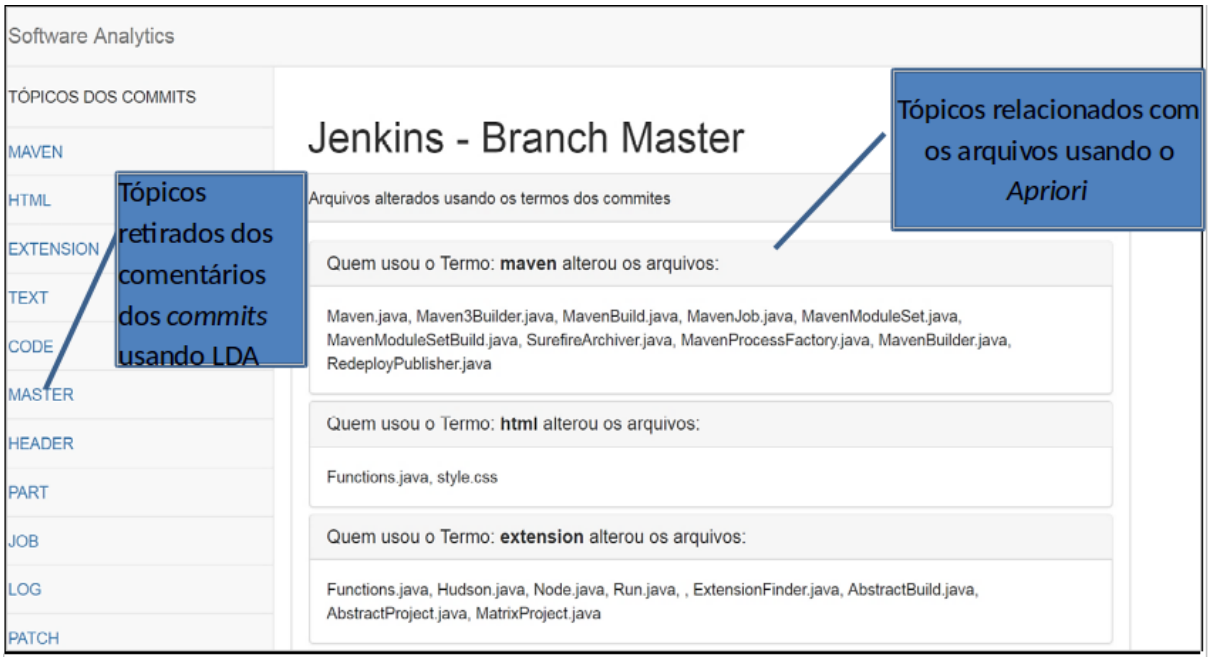
A partir dos tópicos obtidos pelo LDA, foram procurados os arquivos alterados pelos desenvolvedores que usaram esses tópicos nos comentários. Dessa maneira, os desenvolvedores que se referiram a determinado termo alteraram os arquivos encontrados pela regra de associação. A execução desse algoritmo contou com o suporte mínimo inicial de 20%, baixando seus valores para 15% e 10%, que obteve resultados mais satisfatórios para essa pesquisa. O suporte mínimo é o percentual mínimo de vezes que o termo é encontrado na transação. Nesse caso, o tópico foi encontrado em pelo menos 10% das transações com determinado tópico. Quanto

mais alto é valor do suporte mínimo, menor é o valor das relações encontradas. Assim, com suporte mínimo entre 15% e 20%, a quantidade de arquivos encontrados não foi expressiva. Contudo, com 10%, já foi possível identificar arquivos relacionados aos tópicos que tornassem possível entender melhor os tipos de alterações realizadas. Os arquivos *changelog* e pom.xml foram indicados com as maiores alterações. Contudo, por se tratar de arquivos de configuração, é esperado que todas as alterações realizadas no sistema também alterem esses arquivos. Assim, neste estudo julgou-se por bem retirá-los das análises.

Para enriquecer este trabalho, foram analisados, também: a quantidade de tipos ações realizadas nos *commits* por ano (arquivos modificação, adicionados, renomeados e deletados); a quantidade de *commits* por ano; a quantidade de desenvolvedores por ano; os arquivos modificados por desenvolvedor e a quantidade de *commits* por dia da semana. Para analisar esses dados, foi utilizada a ferramenta MicroStrategy Analytics Desktop (MICROSTRATEGY, 2018) na versão *free*.

Após a fase de análise, foi desenvolvido um protótipo da aplicação com os dados do sistema Jenkins. Esse protótipo pode ser visto na Figura 2.

Figura 2 – Protótipo do Sistema de Inteligência Analítica no repositório de código fonte - Relação dos tópicos com a alteração de arquivos



Fonte: Elaborada pelos autores.

De acordo com as análises, foram obtidos tópicos como: *translation* relacionados com o arquivo Messages_fr.properties, indicando o esforço na tradução da equipe do Jenkins para o francês e o comportamento de internacionalização. O mesmo acontece com o termo *Localization* alterando o Message.properties. O termo *Compatibility* foi relacionado aos arquivos: Hudson.java, Mailer.java, XStream2.java, style.css, hudson-behavior.js, Cause.java, AbstractProject.java, ou seja, arquivos que tratam de chamadas de API's e bibliotecas como e-mail e leitura e gravação de XML. O termo *Plugin* foi encontrado na análise e relacionado aos arqui-

vos: PluginManager.java, PluginWrapper.java, ClassicPluginStrategy.java. Isso se justifica pelo fato do Jenkins ser preparado para suportar *plugins* desenvolvidos pela comunidade.

O resultado apresentado torna possível mapear o sistema sem ter a necessidade de grandes investigações no código fonte, facilitando o entendimento do que de fato está sendo feito no projeto. Por exemplo, é possível verificar os assuntos tratados como: segurança, configuração de *build*, regionalização, problemas de regressão de código entre outras correções e saber quais arquivos são afetados pelos seus respectivos assuntos. O Quadro 2 exibe os termos encontrados nos comentários dos *commits* e os arquivos relacionados nesses comentários, juntamente com o exemplo dos tópicos encontrados.

Quadro 2 – Arquivo alterado usando o termo com exemplo dos tópicos encontrados

Termo com os arquivos alterados	Exemplos de Tópicos encontrados
Quem usou o termo **security**, alterou os arquivos: layout.jelly, AbstractProject.java HudsonPrivateSecurityRealm.java	- security security-stable security-rc null stable npe - security security-stable security-rc null stable returns - security security-stable null security-rc stable problem
Quem usou o termo **maven**, alterou os arquivos: Maven.java, Maven3Builder.java, MavenBuild.java, MavenJob.java, MavenModuleSet.java, MavenModuleSetBuild.java, SurefireArchiver.java, RedeployPublisher.java	- maven directory windows files case issue - maven windows directory files jdk issue
Quem usou o termo **revisions**, alterou os arquivos: Descriptor.java, Project.java, Slave.java, AbstractProjectTest.java, Mailer.java, RobustCollectionConverter.java, XStream2.java, Cause.java, MavenJob.java, AbstractProject.java, OldDataMonitor.java, GlobalMatrixAuthorizationStrategy.java, ParametersAction.java, AuthorizationMatrixProperty.java	- revisions svnmerge merge noting tracking cli - revisions svnmerge merge noting tracking command
Quem usou o termo **compatibility**, alterou os arquivos: Hudson.java, Mailer.java, XStream2.java, style.css, hudson-behavior.js, Cause.java, AbstractProject.java	- compatibility back type data change backward - compatibility back type backward data avoid

Quem usou o termo **prepare**, alterou os arquivos: PluginManager.java, HudsonTestCase.java	- prepare development iteration trunk snapshot oss
Quem usou o termo **commit**, alterou os arquivos: Jenkins.java	- commit picked reverts previous ceb fdf
Quem usou o termo **tests**, alterou os arquivos: TestResult.java	- commit picked reverts previous ceb fdf - tests unit system property maven execution - tests unit system property check windows
Quem usou o termo **failed**, alterou os arquivos: AbstractProject.java	- failed class exception stream serialize problem - failed class stream serialize exception problem - failed class serialize exception stream java.lang.runtimeexception
Quem usou o termo **translation**, alterou os arquivos: Messages_fr.properties	- translation french translations class location localization
Quem usou o termo **extension**, alterou os arquivos: Functions.java, Hudson.java, Node.java, Run.java, ExtensionFinder.java, AbstractBuild.java, AbstractProject.java, MatrixProject.java	- extension point class failed implementation serialize - extension point class failed plugins serialize - extension point class failed serialize plugins
Quem usou o termo **localization**, alterou os arquivos: Messages.properties, layout_ja.properties, sidepanel_ja.properties, queue_ja.properties, Messages_ja.properties, Messages_ja.properties, Messages_ja.properties, sidepanel_fr.properties, queue_fr.properties	- japanese localization javadoc resources translation german

Fonte: Elaborado pelos autores.

5 GRUPO FOCAL

Com a finalidade de avaliar o método utilizado neste trabalho, foi realizado um grupo focal com desenvolvedores e gestores de projetos de *software* em uma empresa de tecnologia da

informação de grande porte do estado de Minas Gerais. O grupo focal é um método qualitativo que auxilia na validação de trabalhos empíricos. Trata-se de uma discussão planejada com o propósito de se obter a percepção de determinado grupo sobre um assunto. Em geral, é realizada com 3 a 12 participantes e guiada por um moderador, que mantém o foco da discussão (KONTIO et al., 2008) (KONTIO et al., 2004). A condução do grupo focal deste trabalho segue os seguintes passos (KONTIO et al., 2008, 2004):

Definição do problema - por meio de um protótipo desenvolvido com as técnicas de inteligência analítica descritas neste trabalho, o grupo focal teve como intenção avaliar se o método permite aos engenheiros de *software* descobrir quais arquivos de código fonte do sistema têm sido alterados e quais os motivos de suas alterações. Para isso, procurou-se entender a visão de desenvolvedores e gerentes sobre questionamentos recorrentes no desenvolvimento inspirados pela literatura em alguns trabalhos (BUSE; ZIMMERMANN, 2012) (CAPILUPPI; IZQUIERDO-CORTÁZAR, 2013) (ROBLES et al., 2014). A apresentação do protótipo ao grupo focal teve a intenção de identificar os pontos fortes e fracos do método, e a aplicação de inteligência analítica no repositório de *software*.

5.1 Caracterização dos Participantes

O grupo focal contou com a participação de 6 pessoas, a média de idade foi de 35 anos; o tempo de experiência variou entre 7 a 25 anos na área de desenvolvimento de sistemas; todos graduaram-se em Ciência da Computação ou Sistemas de Informação. Haviam dois mestres, um doutorando e dois pós-graduandos. Entre as certificações, houve um Scrum Master, um Java Programming, um com as certificações PMP, ITIL e MPS.BR e um com certificação Caché ObjectScript.

5.2 Discussões do Grupo Focal

Nesta Seção, são apresentados os questionamentos referentes aos problemas encontrados na Engenharia de *Software* e os comentários do grupo focal. As questões foram tratadas antes e depois da apresentação do protótipo implementado, utilizando a proposta do método descrita na Seção 3. As questões foram tratadas em duas partes, a primeira parte procurou compreender os problemas enfrentados pelos engenheiros de *software* na prática e como eles solucionam estes problemas. A segunda parte exibiu o protótipo da ferramenta implementada neste trabalho e avaliou se os problemas levantados foram solucionados pela ferramenta. A discussão do grupo focal pode ser vista no Quadro 3.

Quadro 3 - Discussão do Grupo Focal

Discussão sobre os problemas na Engenharia de *Software* sem o uso do protótipo da ferramenta proposta neste trabalho.	**Discussão sobre a solução dos problemas na Engenharia de *Software* após conhecerem o protótipo da ferramenta proposta neste trabalho.**
Q1 - Como você consegue uma visão sobre o que de fato está acontecendo com o projeto? Os participantes responderam que costumam obter a visão do projeto por meio de conversas com a equipe, revisão de código, documentação e de ferramentas como Rational Team Concert da IBM. Apesar das ferramentas existentes para obter uma visão do projeto, segundo um participante, "nada melhor do que conversar com as pessoas para saber o que está acontecendo". "Tem coisa que vem da cabeça do desenvolvedor e do analista que está só lá", completa.	Q1 - Com o protótipo apresentado foi possível ter uma visão do projeto? A maioria dos participantes responderam de forma afirmativa, porém algumas visões poderiam ser acrescentadas, como mais visões arquiteturais e gerenciais. Identifica-se que o protótipo permite analisar a evolução do *software* e a contribuição dos desenvolvedores.
Q2 - Quais indicadores ou métricas você utiliza para analisar o projeto de *software*? Foram listados: ponto de função, quantidade de pessoas alocadas no projeto, quantidade de defeitos ou funcionalidades, quantidade de casos de uso, tempo de disponibilização de uma versão para outra, conjunto de métricas CK e horas trabalhadas no projeto. Porém, um participante revela: "não tenho segurança em meus indicadores". Já que muitos dados são colhidos manualmente e nem sempre são atualizados.	Q2- Permite analisar o projeto por meio de indicadores ou métricas? Os participantes afirmaram que foi possível analisar indicadores e métricas de desenvolvimento de forma quantitativa e qualitativa. Sugere-se que o sistema seja parametrizado, com indicadores e métricas fornecidos pelo usuário, bem como a classificação das entregas como corretiva ou evolutiva.

Q3- Em quais situações você utiliza sua experiência ou "intuição"para tomar decisões sobre o projeto de *software*? Os participantes revelam que em mais de 90% de suas tarefas eles sempre utilizam a experiência ou a intuição para tomar decisões. Apesar dos dados obtidos nos projetos de *software*, eles não devem ser usados sozinhos, sendo que a experiência do profissional minimiza o risco de uma decisão malfeita, mesmo de posse de informações sobre o projeto. Ressaltou-se, ainda, a dificuldade para obter informações sobre o sistema durante seu desenvolvimento e manutenção. Em suma, a experiência é geralmente utilizada para planejar projetos, principalmente em projetos muito grandes, em que se tem dificuldade para gerenciar o processo e requisitos, ou até mesmo, durante uma proposta comercial. Um dos participantes afirma que se tem “a certeza que vamos errar no planejamento". Porém, "se você diminui o escopo, tem-se a certeza que vai errar menos", aconselha.	Q3- Permite embasamento para tomada de decisão? Os participantes apontaram que poderia auxiliar nas decisões como: momentos para se fazer builds da aplicação, iniciar testes, quais arquivos merecem maiores atenção na refatoração e decisões sobre o gerenciamento da equipe.
Q4- Como você analisa a evolução do sistema que está sendo desenvolvido ou mantido? Os participantes obtêm essa visão de forma externa, de acordo com as solicitações feitas pelo cliente, o uso do sistema pelos clientes, a quantidade de incidentes e as melhorias solicitadas, assim como acumulação de módulos e a complexidade acidental.	Q4- Permite analisar a evolução do sistema? Os participantes comentaram que o protótipo exibido pode ser útil para avaliar a maturidade e estabilidade do *software*, por meio dos tipos de alterações.

Q5 - Como você mede o seu esforço ou da equipe de desenvolvimento? Foi consenso entre os participantes a utilização de horas para medir o esforço da equipe, ou seja, quantas horas são gastas para cada entrega. Um dos participantes mencionou que utiliza dados mais qualitativos, como as siglas P-M-G (Pequeno, Médio e Grande) para medir os esforços. Então, uma tarefa com P equivale a tarefas que consomem em até quatro horas; M, até seis horas; e G, até nove horas.	Q5- É possível verificar o esforço despendido pelo,desenvolvimento? Este, ponto foi elencado como negativo, pois os commits, apresentados somente exibem suas entregas, não sabendo exatamente,as horas gastas com a mudança ou sua complexidade. O sistema,somente exibiu as contribuições feitas numa visão macro, ou,seja, somente as entregas dos commits, sem outras informações sobre o esforço e o tempo gasto para,cada funcionalidade ou manutenção.
Q6 - Quais ferramentas você utiliza para gerar conhecimento sobre o projeto de *software*? Foram relacionados os programas típicos do desenvolvimento de sistemas, como: Enterprise Architec, Wiki, Microsoft Project e Ganter, assim como repositórios de documentos e o próprio manual do sistema.	Q6 - Agrega conhecimento ao projeto? Neste quesito, o protótipo permite verificar quem trabalhou em cada item e o que tem sido trabalhado. Um dos participantes ressaltou que permite ver o projeto por dentro e conhecer como estão suas mudanças, sendo útil para quem está no projeto compreender melhor o sistema.

Fonte: Elaborado pelos autores.

6 RESULTADOS DO GRUPO FOCAL

O grupo focal teve como finalidade descobrir se o método proposto neste trabalho permite aos engenheiros de *software* obterem informações sobre o projeto de *software*. Por meio do protótipo construído, foi possível ter uma visão da prática de desenvolvimento, avaliar o projeto por meio de indicadores e métricas, analisar a evolução do sistema e servir como ferramenta para gerar conhecimento sobre o projeto de *software*.

De acordo com a entrevista do grupo focal, é possível confirmar que, com as informações geradas pelo protótipo, o grupo focal foi capaz de se engajar rapidamente no projeto, entendendo os marcos de sua evolução. Como exemplo, podem ser citadas atividades como entender as correções feitas no sistema e quais os principais arquivos que foram alterados por essas correções, conhecer os desenvolvedores do Jenkins e saber quais partes do sistema eles

mais atuam, além de seus hábitos de *commits*.

Conforme a discussão, é interessante pontuar que a combinação de análise de dados do projeto com o uso de metodologias ágeis pode tornar projetos mais fáceis de serem conduzidos, visto que os participantes do grupo focal consideraram que a conversa com a equipe é importante para compreender o projeto. Apesar de a inteligência analítica auxiliar o encontro de informações a respeito do projeto de *software*, o engenheiro de *software* não deve dispensar o uso de sua experiência e intuição na tomada de decisão. A inteligência analítica e a experiência devem ser utilizadas em conjunto para maior assertividade em suas decisões.

Entende-se que o protótipo implementado pode ser utilizado como ferramenta para se obter informações do projeto de *software* e expor as alterações no sistema. Permite, ainda, entender as mudanças realizadas no sistema por meio dos *commits*. Assim como o método proposto neste trabalho, é indicado que sejam feitas adaptações para sua implantação em outros projetos.

7 LIMITAÇÕES DO TRABALHO

Não foi possível medir o esforço da equipe de desenvolvimento em sua totalidade. Ressalta-se que, para a realização de análises mais produtivas, é essencial que se tenha o contexto do projeto que será analisado. Apesar de, neste estudo, não ter sido possível uma análise em sistema desenvolvido na empresa dos participantes do grupo focal, esses conseguiram entender o sistema no qual foi realizado o estudo de caso, o Jenkins.

O protótipo implementado não forneceu informações mais gerenciais, nem explorou as métricas de produto como, Loc, complexidade ciclomática e métricas CK, que poderiam agregar mais informações ao projeto. Apesar disso, entende-se que outros trabalhos têm suprido esse tipo de pesquisa (SOKOL et al., 2013), e já estão presentes em ferramentas de mercado como o Sonar (SONARQUBE, 2008). A classificação dos *commits* como corretivas ou melhorias poderia facilitar as análises e revelar novos conhecimentos, assim como a análise dos *commits* por versão.

8 CONCLUSÃO

Este trabalho teve o objetivo de propor um método que auxilie engenheiros de *software* a entenderem melhor o projeto em que estão; e assim procurar responder, se é possível utilizar a inteligência analítica para descobrir quais arquivos de código fonte do sistema têm sido alterados, e quais os motivos de suas alterações. Foi apresentado um método que utiliza técnicas de modelo de tópicos e regra de associação que permitem expressar de maneira textual as alterações e acontecimentos no desenvolvimento do sistema, explorando seu repositório de código fonte.

Para avaliar a utilização desse método, foi implementado um protótipo, tendo como exemplo os dados do sistema de código aberto Jenkins. Por meio de um grupo focal, foi possível verificar que o protótipo apresentado permite que sejam entendidas as mudanças do *software*, sem a necessidade de fazer uma investigação profunda no código. Essas informações permitem aos engenheiros de *software* tomarem decisões baseadas em dados do projeto em relação à refatoração, aos *builds*, ao planejamento de testes e ao gerenciamento da equipe. Um ponto negativo do protótipo foi não exibir o tempo de esforço gasto pelo desenvolvedor ao realizar uma tarefa.

Os métodos apresentados neste trabalho podem ser automatizados para geração de relatórios agregados ao repositório de *software*, seja por meio de um *dashboard*, ou por meio de um *plugin* que seja uma extensão do ambiente de desenvolvimento e do sistema de controle de versão. Um exemplo do protótipo com os dados do repositório do Jenkins pode ser visto no link: <http://bruno-rodrigues.dx.am/pages/index.php>. Com a finalidade de promover novos estudos nesse contexto, foram disponibilizados os dados dos *commits* do Jenkins usados neste trabalho, os quais podem ser encontrados no link: <https://github.com/brunorodriguesti/scm_jenkins> e estão no formato do banco de dados MySQL. Como trabalhos futuros, pretende-se criar uma ferramenta de *dashboard*, adicionando visões de esforço da equipe e classificações das manutenções.

REFERÊNCIAS

AGRAWAL, Rakesh; SRIKANT, Ramakrishnan et al. Fast algorithms for mining association rules. In: **Proc. 20th int. conf. very large data bases, VLDB**. [S.l.: s.n.], 1994. v. 1215, p. 487–499. Acesso em: 09 abr. 2016.

ALALI, Abdulkareem; KAGDI, Huzefa; MALETIC, Jonathan I. What's a typical commit? a characterization of open source software repositories. In: IEEE. **Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on**. [S.l.], 2008. p. 182–191.

BARUA, Anton; THOMAS, Stephen W; HASSAN, Ahmed E. What are developers talking about? an analysis of topics and trends in stack overflow. **Empirical Software Engineering**, Springer, v. 19, n. 3, p. 619–654, 2014.

BUSE, Raymond PL; ZIMMERMANN, Thomas. Information needs for software development analytics. In: IEEE PRESS. **Proceedings of the 34th international conference on software engineering**. [S.l.], 2012. p. 987–996.

CAPILUPPI, Andrea; IZQUIERDO-CORTÁZAR, Daniel. Effort estimation of floss projects: a study of the linux kernel. **Empirical Software Engineering**, Springer, v. 18, n. 1, p. 60–88, 2013.

DIT, Bogdan et al. Impactminer: A tool for change impact analysis. In: ACM. **Companion Proceedings of the 36th International Conference on Software Engineering**. [S.l.], 2014. p. 540–543. Acesso em: 16 nov. 2014.

FOURNIER-VIGER, Philippe et al. Spmf: a java open-source pattern mining library. **The Journal of Machine Learning Research**, JMLR. org, v. 15, n. 1, p. 3389–3393, 2014.

HASSAN, Ahmed E. The road ahead for mining software repositories. In: IEEE. **Frontiers of Software Maintenance, 2008. FoSM 2008.** [S.l.], 2008. p. 48–57.

HINDLE, Abram et al. Automated topic naming to support cross-project analysis of software maintenance activities. In: ACM. **Proceedings of the 8th Working Conference on Mining Software Repositories**. [S.l.], 2011. p. 163–172. Acesso em: 10 jul. 2017.

HSU, Sheng-Kuei; LIN, Shi-Jen. Macs: Mining api code snippets for code reuse. **Expert Systems with Applications**, Elsevier, v. 38, n. 6, p. 7291–7301, 2011.

HU, Jiajun et al. Modeling the evolution of development topics using dynamic topic models. In: IEEE. **Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on**. [S.l.], 2015. p. 3–12.

JENKINS. 2011. Disponível em: <https://jenkins.io/index.html>. Acesso em: 02 mai. 2017.

KONTIO, Jyrki; BRAGGE, Johanna; LEHTOLA, Laura. The focus group method as an empirical tool in software engineering. In: **Guide to advanced empirical software engineering**. [S.l.]: Springer, 2008. p. 93–116.

KONTIO, Jyrki; LEHTOLA, Laura; BRAGGE, Johanna. Using the focus group method in software engineering: obtaining practitioner and user experiences. In: IEEE. **Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on**. [S.l.], 2004. p. 271–280.

MCCALLUM, Andrew Kachites. Mallet: A machine learning for language toolkit. Http://mallet.cs.umass.edu. 2002.

MICROSTRATEGY. **Powerful Data Analytics & Visualization Tools**. 2018. Disponível em: <https://microstrategy.com/us/home-8>.

ROBLES, Gregorio et al. Estimating development effort in free/open source software projects by mining software repositories: a case study of openstack. In: ACM. **Proceedings of the 11th Working Conference on Mining Software Repositories**. [S.l.], 2014. p. 222–231. Acesso em: 16 nov. 2014.

SOKOL, Francisco Zigmund; ANICHE, Mauricio Finavaro; GEROSA, Marco Aurélio. Metricminer: Supporting researchers in mining software repositories. In: IEEE. **Source Code Analysis and Manipulation (SCAM), 2013 IEEE 13th International Working Conference on**. [S.l.], 2013. p. 142–146.

SONARQUBE. **Continuous Code Quality | SonarQube**. 2008. Disponível em: <https://www.sonarqube.org/>. Acesso em: 10 fev. 2017.

STEYVERS, Mark; GRIFFITHS, Tom. Probabilistic topic models. **Handbook of latent semantic analysis**, v. 427, n. 7, p. 424–440, 2007.

THOMAS, Stephen W. Mining software repositories using topic models. In: ACM. **Proceedings of the 33rd International Conference on Software Engineering**. [S.l.], 2011. p. 1138–1139. Acesso em: 15 nov. 2014.

ZHANG, Dongmei et al. Software analytics as a learning case in practice: Approaches and experiences. In: ACM. **Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering**. [S.l.], 2011. p. 55–58. Acesso em: 15 nov. 2014.

ZIMMERMANN, Thomas et al. Mining version histories to guide software changes. **IEEE Transactions on Software Engineering**, IEEE, v. 31, n. 6, p. 429–445, 2005.