EXTRAÇÃO DE CONHECIMENTO A PARTIR DE REGRAS DE ASSOCIAÇÃO ENTRE MÉTRICAS DE CÓDIGO FONTE

João Luiz Ramalheira de Almeida, Renato Balancieri, Max Naegeler Roecker, Gislaine Camila Lapasini Lea
Universidade Estadual de Maringá
jlramalheira@gmail.com, renatobalancieri@gmail.com, max.roecker@gmail.com, gclleal@uem.br

Paulo Henrique Bermejo
Universidade de Brasília
paulobermejo@unb.br

Abstract: **Following and register all the produced artifacts along the software development with the source code metrics and commits messages can be a hard task as the software grows in size and complexity. Data Mining tools, such as the *Knowledge Discovery in Database* (KDD), can be a helpful resource to detect patterns, characteristics and aspects of the development process and team. This paper presents the use of Association Rules in source code metrics with the goal to extract knowledge of source code repositories to identify important features in software's development. A model based on KDD described and a prototype implementing this model was developed. The prototype is characterized as a primary study relative to the application of the model in an example. This study was conducted aiming to characterize the use of the model in a specific context and serves as proof of concept. Various Apache Foundation's projects evaluated to extract generalizable patterns of the developers and the impacts in the software product. Based on the outcomes of this tool, project managers can easily identify when the development process is in unwanted way and decide new strategies to put it on the right way. With this, it is concluded that knowledge extraction in source code repositories can be a helpful tool to support the decision-making on the software development.**

***Keywords*: Software Engineering; Knowledge engineering in software; Software development process management; Source code metrics.**

I. INTRODUÇÃO

O processo de produção de um software pode ser dividido em três etapas: definição, desenvolvimento e manutenção [1]. A última etapa, a manutenção, é crítica e seus custos correspondem de 50 a 90% do custo do projeto [2]. Quanto maior a qualidade do software, menor o número de manutenções corretivas necessárias, aumentando a evolução do mesmo [3].

Para melhorar a qualidade de um produto é fundamental que o processo de produção do mesmo seja monitorado [4]. As métricas são utilizadas constantemente para descrever a qualidade de objetos [3] e são importantes para qualquer disciplina da Engenharia de Software [1].

Dessa forma, é necessário que o gerente do projeto acompanhe a evolução do software e seja capaz de intervir antes que um problema possa ocorrer. Além disso, é de suma importância conhecer cada membro da sua equipe, para que possa assumir novos desafios e novos projetos. Porém, compreender essa evolução é uma tarefa árdua [5]. Para conseguir captar essa evolução é necessário realizar um registro de todos os artefatos gerados durante o processo de desenvolvimento do software, para essa finalidade pode-se utilizar um Sistema de Controle de Versão (SCV) [6], também conhecidos como repositórios de código fonte.

O volume de artefatos produzidos durante o desenvolvimento de um projeto de software é elevado [7]. Dessa forma, é importante que os dados sejam filtrados de maneira que possam produzir alguma informação útil para um gerente de projeto de software.

Para essa finalidade a utilização de técnicas de mineração de dados pode ser uma boa estratégia, visto que possuem como principal objetivo extrair informações de bases de dados, viabilizando melhor conhecimento dos dados para dar suporte a tomada de decisão [8].

Estudos desenvolvidos nessa área revelam informação útil ao desenvolvimento de um projeto [5], além de possibilitar a identificação de padrões que podem ser generalizáveis para outros softwares. Essas técnicas podem ser utilizadas para encontrar mudanças que introduzam erros no software [9], para descobrir quais artefatos devem ser modificados quando ocorre alguma alteração no software [10], para análise de redes sociais entre desenvolvedores e para análise de tendências e *hotspots* [11].

O presente artigo está organizado em quatro seções além desta introdutória. Na Seção II tem-se a fundamentação teórica. Na Seção III é descrito o desenvolvimento do modelo baseado no processo KDD. A Seção IV apresenta os resultados obtidos. Por fim, na Seção V são realizadas as considerações finais, destacando as contribuições e trabalhos futuros.

II. FUNDAMENTAÇÃO TEÓRICA

O processo de Descoberta de Conhecimento, do inglês, *Knowledge Discovery in Database* (KDD) [12], possui como

objetivo principal extrair informações de uma base de dados, viabilizando melhor conhecimento para dar suporte à tomada de decisão [8]. Esse processo envolve as seguintes etapas:

- Seleção: primeira etapa do sistema onde se escolhe um conjunto de dados que se deseja explorar;
- Pré-processamento: como os dados podem possuir diversos formatos e vir de diferentes formas é necessário um processamento para padronizá-los, verificando, também, dados redundantes e inconsistentes;
- Transformação: os dados necessitam ser armazenados e formatados de acordo com os algoritmos que serão executados;
- Mineração de Dados: consiste na exploração e análise dos dados transformados, esse processo pode ser automático ou semiautomático. Essa etapa normalmente é a mais custosa de todo o processo, nela são definidas as tarefas e as técnicas que serão utilizadas;
- Análise: nesta etapa é realizada a análise dos dados fornecidos pela etapa anterior, podendo ser realizada de forma manual ou então utilizar técnicas de Inteligência Artificial (IA).

A tarefa no processo de mineração de dados consiste na especificação do que se quer extrair dos dados, diferente das técnicas que são métodos que dão suporte para descobrir os padrões que se quer extrair [13].

Há diferentes tarefas de mineração de dados, entre elas: Regras de Associação, Padrões Sequenciais, Modelos de Classificação, Agrupamento dos Dados, entre outros [13].

As Regras de Associação mostram relações que existem entre itens no domínio que possuem ocorrência significativa. As regras de associação são determinadas com um certo grau de certeza, que podem ser definidos por dois índices: fator de suporte e fator de confiança [9]. A utilização de Regras de Associação no contexto de Mineração de Dados em repositórios de código fonte traz bons resultados [3][5][14].

O processo de Regra de Associação é formalizado da seguinte maneira: Seja $I = \{I_1, I_2, I_3, \cdots, I_m\}$ um conjunto de atributos binários chamados de itens e seja T uma base de dados de transações onde cada t é representada por um vetor binário, com $t[k] = 1$ se t indica a compra do item I_k e $t[k] = 0$, caso contrário. Existe uma tupla na base de dados para cada transação. Seja X um conjunto de itens em I. É dito que a transação t satisfaz X se, para todos itens I_k em X, $t[k] = 1$. Uma regra de associação é uma implicação da forma $X \Rightarrow Y$, onde $X \subset I$, $Y \subset I$ e $X \cap Y = \emptyset$. A regra $X \Rightarrow Y$ é válida no conjunto de transações T, com grau de confiança c, se $c\%$ das transações em T que contêm X também contêm Y. A regra $X \Rightarrow Y$ tem suporte s em T, se $s\%$ das transações em T contêm $X \cup Y$. Se as condições forem satisfeitas, $c\%$ representará o fator de confiança e $s\%$ o fator de suporte [15][16].

As técnicas de Mineração de Dados são divididas em: aprendizado supervisionado (preditivo) e não supervisionado (descritivo) [17]. As técnicas não-supervisionadas não precisam de uma pré-categorização dos dados, ou seja, não é necessário um atributo alvo [18]. As técnicas supervisionadas necessitam de uma especificação para cada exemplo.

Dentre as diversas técnicas destaca-se: Redes Neurais, Árvore de Decisão, Lógica Nebulosa, Estatística e Apriori.

A técnica Apriori é a mais conhecida para Mineração por Regras de Associação [18] e formalmente explicada por meio do algoritmo Apriori, que possui três fases principais: geração, poda e validação [13]. A fase de geração e poda acontece na memória principal, sem a necessidade de vasculhar toda a base de dados utilizada. Nestas etapas são realizadas uma busca em profundidade gerando conjunto de itens candidatos e a poda dos conjuntos que não são tão frequentes na base de dados. Por fim, é realizada uma validação dos conjuntos restantes, verificando se eles atendem a confiança e suporte desejados.

As métricas de código fonte e Mineração de Dados podem ser utilizadas conjuntamente com a finalidade de ajudar a Engenharia de Software a melhorar a qualidade do produto envolvido [3]. Essas métricas são utilizadas para mensurar propriedades de um sistema que será analisado [5].

As métricas podem ser mapeadas em Simples e Compostas [3]. As métricas simples são as que podem ser extraídas diretamente do código fonte. Já as compostas são calculadas a partir das simples ou até mesmo de outras compostas. As métricas utilizadas nesse experimento são:

- Simples:
  a. Complexidade Ciclomática (ACC): Mede o número de caminhos linearmente independentes de um módulo do programa [19]. Essa métrica está diretamente relacionada à facilidade de manutenção [20];
  b. Número de Métodos (NOM): contabiliza a quantidade de métodos do software;
  c. Linhas de Código (LOC): Efetua a contagem da quantidade de linhas de códigos existentes no código fonte do software [21];
  d. Métricas de Acesso a Dados (DAM): contabiliza a razão entre a quantidade de métodos privados e o total de métodos da Classe. Para utilizar essa métrica é realizada uma média de todas as classes do projeto;
  e. Tamanho da Interface da Classe (CIS): contabiliza a quantidade de métodos públicos de uma classe, para tal é calculada a média de todas as classes do projeto.
- Compostas:
  f. Densidade de Linhas por método: Calcula a quantidade média de linhas por método, essa métrica é composta pois utiliza valores extraídos de LOC (linhas de código) e NOM (número de

métodos). Essa métrica é definida pela seguinte expressão: $\frac{LOC}{NOM}$.

III. MÉTODO E DESENVOLVIMENTO

A elaboração do modelo para extração de conhecimento em repositórios de código fonte foi pautada nos passos do KDD: Seleção, Pré-Processamento, Transformação, Mineração de Dados e Análise [12].

Na fase de **Seleção** foram definidos os atributos a serem coletados da base de dados. A Fig.1 ilustra o diagrama de classes dessa etapa, contendo todos os dados coletados. A Classe Project além de um identificador único (id), armazena o nome do projeto (name), a URI (*Uniform Resource Identifie*r) e o caminho que se encontra o projeto no servidor *(absolutePath*). A Classe Commit contém um identificador único (id), o código sha (*secure hash algorithm*) que define unicamente o *commit* no projeto, informações do autor (author), data (date) e mensagem utilizada (message). Por fim, a Classe File contém um identificador único, as informações das métricas utilizadas nesse trabalho, além do caminho do arquivo no servidor (*path*). Todos esses dados são selecionados e extraídos do Projeto.

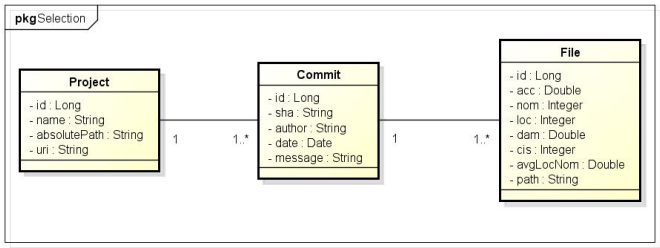

Figura 1. Diagrama de classes da fase de seleção.

Na fase de **Pré-processamento** foi realizada a padronização dos dados, retirando os dados inconsistentes. As métricas extraídas na etapa de seleção são medidas absolutas das classes e não medem alterações realizadas ao longo da evolução do sistema.

Para obter uma medida que avalie o quanto o software evoluiu com uma nova versão é realizado um cálculo delta para cada métrica de software. Esse delta corresponde à diferença entre o valor atual e o valor anterior. Entende-se por valor anterior a última vez que houve alguma alteração em determinado arquivo (não o valor da última versão em si). Isso, porque um arquivo pode ser criado em uma versão e ser modificado somente depois de várias versões. Quando um arquivo é adicionado ao repositório, o valor do delta é o valor absoluto da métrica e quando um arquivo é removido, o valor delta é o oposto da sua última análise.

Além dos valores delta das métricas de código fonte, nesta etapa também é analisada qual a característica da revisão, essa pode ser um *commit* de refatoração, ou seja, reescrita ou reestruturação do código para melhorar o desempenho ou a escrita do algoritmo ou pode ser de correção de *bugs* ou outro tipo de *commit*. Essa análise se dá por meio da junção de padrões com a mensagem utilizada pelo desenvolvedor no momento do *commit* [22]. Por fim é extraído o autor da versão e a quantidade de classes que foram alteradas nessa versão. A Fig. 2 exibe a classe que comporta todas as propriedades da fase de pré-processamento.

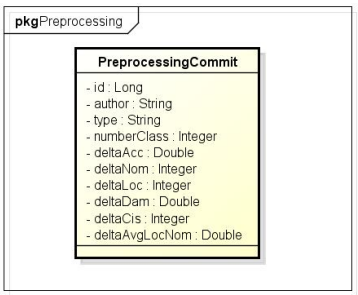

Figura 2. Classe da fase de pré-processamento.

Após a realização da seleção e pré-processamento dos dados, ocorreu a fase de **Transformação**, em que os dados foram formatados adequadamente para aplicar a fase de Mineração de Dados. Devido à natureza do algoritmo Apriori permitir somente atributos categóricos, é necessário realizar a discretização dos dados obtidos [23]. Para os atributos contínuos foi utilizada a Técnica de Bucket, categorização ou utilização de todos os valores como elementos de um conjunto. Nesta técnica são criados diversos baldes com limitantes diferentes. Após a criação desses baldes, cada valor de delta é adequado com a média do balde no qual ele se encaixa.

A técnica de categorização avalia o delta e atribui um valor *Up* para valores de delta maiores que zero, *Down* para valores menores que zero e *Estable* para valores iguais a zero.

Os atributos não contínuos, como autor e tipo do *commit* são utilizados como conjunto.

Na fase de **Mineração de Dados** foi utilizado o algoritmo Apriori, que conta com um conjunto de atributos configuráveis, dentre eles: quantidade de regras, confiança mínima, limites superior e inferior para suporte mínimo. Dessa forma é possível alterar a maneira com que são extraídas as regras de associação.

Ao executar o Apriori, o mesmo retorna um conjunto de regras de associação encontradas para o conjunto de dados utilizados. Essas regras contêm medidas de interesse que são úteis para a interpretação de resultados, esses valores são: suporte, confiança e lift. Suporte é a fração das transações que ocorrem entre os itens precedentes e consequentes. Confiança mede a frequência com que os itens consequentes aparecem nas transações que contêm os precedentes. E lift é o valor da confiança da regra dividido pelo suporte do consequente.

A última fase trata da **Análise**, ou seja, deve-se interpretar as regras de associação extraídas do conjunto de dados para retirar informações importantes entre os atributos, servindo de apoio para tomada de decisão. Cada regra de associação é formada por dois grupos de itens (precedentes e consequentes) e as medidas

de interesse. Para exibição desses dados em formato de gráfico ou tabela é necessário um processamento textual, extraindo das regras alguns dados importantes para serem plotados ou tabelados.

IV. RESULTADOS

Para consolidar e validar o modelo apresentado na Seção III, foi desenvolvido um protótipo em plataforma *Web* que implementa todas as etapas do processo KDD. O protótipo, caracteriza-se como um estudo primário relativo a aplicação do modelo em um exemplo. Esse estudo foi conduzido visando caracterizar o uso do modelo em um contexto específico e serve como prova de conceito [24]. A fonte de dados foi Apache Software Foundation e unidade de análise formada por 10 projetos [25].

O fluxo do programa inicia no cadastro do projeto que será analisado, em seguida os dados são selecionados, pré-processados e transformados para então serem salvos em um arquivo no padrão aceito pelo Weka [26] com suas diretivas e formato específico. Em seguida é aplicada a Mineração de Dados utilizando Regras de Associação, essa execução se dá por meio da utilização da API (*Application Programming Interface*) do Weka. Após a execução do algoritmo, são apresentadas as informações por meio de gráficos e/ou tabelas.

O sistema proposto é baseado no modelo Cliente-Servidor. A comunicação entre o cliente e o servidor é realizada utilizando o protocolo HTTP (*HyperText Transfer Protocol*) em sua versão 1.1. A Fig. 3 destaca a visão arquitetural do sistema.

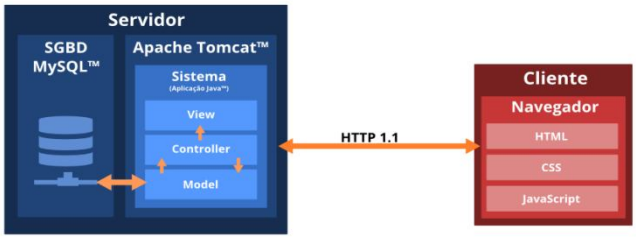

Figura 3. Visão arquitetura do protótipo.

O Servidor foi construído utilizando o modelo MVC (*Model-View-Controller*) que é baseado em camadas. Na camada *Model* estão todas as classes que são persistidas no Sistema de gerenciamento de Banco de Dados (SGBD). A camada *View* contém as interfaces com o usuário da aplicação. A camada *Controller* é responsável por ligar a *View* com a *Model*.

O cliente, por meio do navegador, solicita ou envia informações para um dos controladores, que processa a informação, consulta a camada *Model*, se necessário, e responde ao usuário por meio da camada *View*. Exceto o módulo de visão, todos os demais módulos que serão explanados nas seções seguintes estão implementados na camada *Model*.

A estrutura do Servidor é constituída de duas partes: o SGBD e o servidor da aplicação. Como SGBD utilizou-se o MySQL que é responsável pela persistência dos dados. Já como servidor de aplicação utilizou-se o Apache Tomcat. O Tomcat é responsável por capturar e encaminhar as requisições da rede para suas aplicações, optou-se utilizar o Tomcat devido a sua simplicidade de manutenção em relação as especificações Java EE. Em cada uma das camadas do Modelo MVC, as seguintes tecnologias foram utilizadas:

- *Model:* utilizou-se classes Java para especificação dos dados. A persistência dos dados é feita por meio do DAO (Data Access Object). O DAO utiliza a implementação do EclipeLink e EasyCriteria, dois frameworks que seguem a especificação Java Persistence API (JPA).
- *Controller:* implementada com Java Servlets, que é uma classe de linguagem Java que dinamicamente processa requisições e respostas no servidor da aplicação.
- *View:* implementada com Java Server Pages (JSP), sendo esta uma página HTML (*HyperText Markup Language*) com códigos Java embutidos para fornecer um conteúdo dinâmico.

O Cliente foi desenvolvido utilizando HTML em sua versão 5. O HTML é utilizado com a finalidade de estruturar e organizar informações. Para a apresentação e estilização das informações estruturadas pelo HTML utilizou-se o CSS (*Cascade Style Sheets*), em sua versão 3 por meio da biblioteca Bootstrap. Para uma melhor interação com o usuário, utilizou-se também JavaScript por meio da biblioteca JQuery.

Quando o usuário do sistema preenche o formulário ilustrado na Fig. 4, essa informação é enviada ao Servidor que repassa a solicitação ao módulo de Seleção. No módulo de Seleção é efetuado um clone do projeto e extraído todos os dados definidos no modelo proposto.

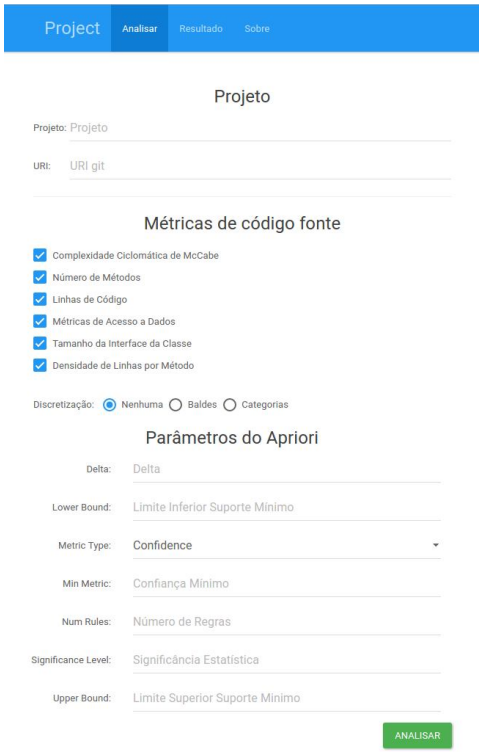

Figura 4. Formulário de cadastro de projeto.

Durante a etapa de pré-processamento são analisados todos os *commits*, verificando os arquivos alterados nesta revisão e calcula-se o delta de cada atributo. Além do cálculo é armazenado um identificador do desenvolvedor (por meio de um apelido único afim de preservar a identidade do mesmo), a quantidade de classes alteradas na revisão e o tipo do *commit*.

O tipo do *commit* é avaliado utilizando a junção de padrões no texto da mensagem escrita pelo desenvolvedor. A Tabela I apresenta alguns padrões utilizados na identificação do *commit*. Os padrões seguem o padrão sugerido pela comunidade de software livre. Nesta junção de padrões é ignorada a diferença de letras maiúsculas e minúsculas (*case insensitive*). O símbolo asterisco (*) é um caractere curinga que significa qualquer quantidade de caracteres ao fim da palavra.

TABELA I
PADRÕES DE IDENTIFICAÇÃO DE *COMMIT*

Identificação	**Padrões**	**Descrição**
bug	*bug* | *fix** | *resolve** | *issue*	Procura na mensagem as palavras *bug*, *fix* (e qualquer derivação dela tais como: *fixed*, *fixes*, *fixing*), *resolve* (e qualquer derivação dela tais como: *resolves*, *resolved*) e *issue*.
refactor	*refactor**	Procura na mensagem a palavra *refactor* e qualquer derivação dela tais como: *refactoring*, *refactorings*.
other	-	Classifica como outro automaticamente quando não é encontrado nenhum padrão pré estabelecido.

Após os dados serem pré-processados, eles são transformados aplicando-se a técnica de discretização dos dados (*buckets* ou categorização), de acordo com parâmetros informados pelo usuário na solicitação da mineração. Caso o usuário deseje, ele pode realizar a Mineração de Dados utilizando todos os valores possíveis para cada atributo.

Assim que a transformação dos dados é concluída, é gerado um arquivo no formato ARFF (*Attribute-Relation File Format*) que será utilizado no módulo de mineração.

O módulo de Mineração de Dados é implementado utilizando o Weka, uma biblioteca desenvolvida em Java que conta com diversos algoritmos de aprendizado de máquina, os algoritmos podem ser rodados utilizando a plataforma gráfica ou executados de uma aplicação Java.

A camada de apresentação foi desenvolvida com o auxílio da biblioteca Google Chats, desenvolvida em JavaScript e HTML 5 para a construção de gráficos. Os gráficos exibidos são: evolução da complexidade Ciclomática de McCabe e *commits* por desenvolvedor. A Fig. 5 mostra a tela de resultados contendo as informações do projeto e os gráficos.

Além dos gráficos que são exibidos na tela, é disponibilizado ao usuário da aplicação a opção de *download* do arquivo ARFF gerado para utilização do Weka e do arquivo de resultados das regras de associação.

Figura 5. Interface de resultados.

Para avaliação do protótipo foram analisados 10 projetos da *Apache Software Foundation*. Ao todo foram processados 113.819 *commits* e 418.724 arquivos. A Tabela II exibe o nome do projeto, a quantidade de *commits* e o total de arquivos processados para cada projeto.

TABELA II
PROJETOS ANALISADOS

Projeto	Total de Commits	Total de Arquivos Processados
Apache OFBiz	23.440	33.278
Apache Tomcat	17.225	42.932
Apache Hadoop	13.839	117.091
Apache Ant	13.342	42.792
Apache Felix	13.031	43.597
Apache CXF	12.058	51.810
Apache Maven	10.319	24.428
Apache Derby	8.111	39.229
Apache Abdera	1.498	6.436
Apache Harmony	956	17.131

Os projetos foram executados com diversos parâmetros diferentes, tais como quantidade de métricas de código fonte e diferentes técnicas de discretização. Para os resultados apresentados, foram utilizados os valores padrões do Weka, sendo eles:

- delta: 1
- lowerBoundMinSupport: 0,1
- metricType: confidence
- minMetric: 0,5
- numRules: 10
- significanceLevel: -1
- upperBoundMinSupport: 1

Para avaliar a aplicação do modelo detalhado em extrair conhecimento de repositórios de código fonte, a Tabela III apresenta três regras: as duas primeiras com características do desenvolvedor e a última com características gerais do projeto. Destas informações duas tem como peculiaridade padrões generalizáveis e uma com impacto de mudanças, conforme apresentado por [11].

TABELA III
REGRAS EXTRAÍDAS

Regra	Confiança
Quando o desenvolvedor 4 do Projeto Apache Derby realiza alteração, então a modificação é para corrigir *bug*.	0.95
Quando o desenvolvedor 2 do Projeto Apache Tomcat realiza alteração, então a modificação é de refatoração.	0.81
Quando o *commit* é do tipo *bug*, a complexidade Ciclomática sobe.	0.56

Utilizando todas as métricas de código fonte e nenhuma técnica de discretização foi possível extrair do projeto Apache Derby uma informação que pode vir a ser importante para o gerente da equipe. O desenvolvedor 4 (omitido seu nome por questões de privacidade) realiza uma grande quantidade de mensagem de correção de *bug*, revelando que tal desenvolvedor pode ser um membro experiente e importante para a equipe. Essa informação pode ser observada na Fig. 6.

```
authors=Desenvolvedor 4 401 ==> type=bug 379
<conf:(0.95)> lift:(1.01) lev:(0) [2] conv:(1.06)
```

Figura 6. Regra extraída do projeto Apache Derby

Ainda com os mesmos parâmetros de execução foi possível extrair do projeto Apache Tomcat a informação de que o Desenvolvedor 2 realiza *commits* de refatoração de código, ilustrado na Fig. 7. Isso pode ser justificado por ele ser o desenvolvedor que realizou mais da metade dos *commits* e, portanto, tem experiência e conhecimento sobre todo o projeto para realizar diversas refatorações. Conhecer os desenvolvedores da equipe e ter ciência de quais são os mais experientes permite a

uma instituição escolher novos líderes e representantes para novos projetos.

```
authors=Desenvolvedor 2 232 ==> type=refactor 188
<conf:(0.81)> lift:(1.12) lev:(0.01) [20] conv:(1.44)
```

Figura 7. Regra extraída do projeto Apache Tomcat

As duas Regras podem ser verificadas por meio dos exemplos exibidos na Fig.8 e Fig. 9, onde são apresentadas mensagens de *commit* do próprio repositório no GitHub.

Figura 8. Gabarito para regra do projeto Apache Derby

Na Fig. 8, é importante destacar que o projeto Apache Derby possui 8.111 *commits*, no qual destes 1.161 são caracterizados como correção de *bugs* além de 27 desenvolvedores contribuindo. Fazer com que o gerente acompanhe todos esses *bugs* resolvidos e ainda conseguir fazer um relacionamento com qual desenvolvedor resolveu determinado problema é uma tarefa extremamente árdua, mas ao mesmo tempo considerável.

Figura 9. Gabarito para regra do projeto Apache Tomcat

Assim como no exemplo anterior, na Fig. 9, existem diversas mensagens que se adequam ao padrão de refatoração realizadas pelo Desenvolvedor 2, porém encontrá-las entre 17.225 mensagens de *commits* pode ser uma tarefa complicada.

Utilizando a técnica de discretização por classificação não foi possível obter sucesso utilizando todas as métricas. A justificativa para tal é que os valores das métricas ficam muito semelhantes, fazendo com que o algoritmo Apriori encontre regras que podem não ser importantes para o usuário. Quando as regras são limitadas a duas, pode-se encontrar resultados satisfatórios, como o exibido na Fig. 10. Nota-se que quando o *commit* é do tipo *bug*, a complexidade ciclomática sobe.

```
type=bug 3394 ==> deltaaac=Up 1903
<conf:(0.56)> lift:(1) lev:(-0) [0] conv:(1)
```

Figura 10. Regra extraída do projeto Hadoop

A justificativa do da terceira regra apresentada se dá baseada na premissa de que uma falha é a falta de conformidade com os requisitos de software [1]. Para resolver o sistema deverá ser corrigido fazendo com que o desenvolvedor que realiza tal correção aumente a complexidade para atender todos os requisitos elicitados.

V. CONSIDERAÇÕES FINAIS

O presente trabalho abordou a utilização de mineração de dados em repositórios de código fonte. O objetivo foi extrair conhecimento destes repositórios detalhando um modelo baseado no KDD que utiliza métricas de código fonte e estabelece regras de associação entre elas. Para validação deste modelo foram realizados diversos experimentos com projetos da Apache e vários parâmetros para o algoritmo Apriori. Utilizou-se como gabarito das regras extraídas a literatura e o próprio histórico do projeto.

Os resultados obtidos com os diversos projetos minerados se mostram satisfatórios, visto que as regras apresentadas podem ser interessantes para apoiar o gerente de projetos na tomada de decisão.

Apesar de conseguir algumas informações importantes, outras são irrelevantes ou possuem um grau de confiança muito baixo, isso porque os dados coletados nem sempre apresentam um padrão, tornando difícil encontrar padrões com alto nível de confiança.

Conclui-se que a extração de conhecimento em repositórios de código fonte utilizando o modelo detalhado é capaz de identificar informações significativas para a evolução do software, permitindo o acompanhamento do projeto de maneira mais rápida e clara, possibilitando ao gerente uma reação mais rápida quando é verificado comportamentos indesejáveis ou que seguem a mesma linha de projetos que já tiveram problemas.

A principal limitação do protótipo desenvolvido se refere a realização de medida das métricas somente em projetos com linguagem de programação Java, seria interessante uma evolução do protótipo para poder avaliar outras linguagens Orientada a Objetos como C++, Python, Ruby, entre outras. Outra limitação se refere ao sistema de controle de versão, sendo suportado somente o Git.

Durante o desenvolvimento deste trabalho foram identificados alguns trabalhos futuros, tais como: i) utilização de outras tarefas e técnicas de mineração de dados, podendo adotar técnicas de classificação com árvore de decisão e redes neurais; ii) utilização de outras métricas de código que não se referem a somente o código fonte, mas sim ao código compilado também;

iii) análise de repositórios de outras instituições, sendo de software livre ou até mesmo entidades privadas; iv) validação do protótipo desenvolvido a partir da metodologia de painel com especialistas [27] (gerentes de projetos); v) criação de uma ferramenta robusta baseada no protótipo para ser disponibilizada como software livre;

REFERÊNCIAS

[1] R. S. Pressman, Engenharia de Software - Uma Abordagem Profissional, 7 ed., Nova Iorque: McGraw-Hill, 2011.

[2] A. April, and A. Abran, A. Software Maintenance Management: Evaluation and Continuous Improvement. ISBN: 978-0-470-14707-8. Wiley-IEEE Computer Society Press, 2008.

[3] D. D. C. Ribeiro, OSTRA: Um estudo do Histórico da Qualidade de Software Através de Regras de Associação de Métricas, Niterói: Universidade Federal Fluminense, 2012.

[4] T. Hall, N. Baddoo, and D. Wilson, “Measurement in Software Process Improvement Programmes: An Empirical Study,” New Approaches in Software Measurement,pp. 73-82, Springer, 2001.

[5]F. Sokol, MetricMiner: uma ferramenta web de apoio à mineração de repositórios de software, São Paulo: Universidade de São Paulo, 2012.

[6] S. Li, H. Tsukiji and K. Takano, "Analysis of Software Developer Activity on a Distributed Version Control System," 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA), Crans-Montana, 2016, pp. 701-707.

[7] S. P. Reiss, "Incremental Maintenance of Software Artifacts," in *IEEE Transactions on Software Engineering*, vol. 32, no. 9, pp. 682-697, Sept. 2006. doi: 10.1109/TSE.2006.91

[8] J. Han, M. Kamber and J. Pei, Data Mining: Concepts and Techniques, 3 ed., Burlington, Massachusetts: Morgan Kaufmann, 2011, p. 744.

[9] S. Kim, T. Zimmermann, K. Pan e E. J. Whitehead Jr., “Automatic Identification of Bug-Introducing Changes,” em 21st IEEE/ACM International Conference on Automated Software Engineering, Washington, 2006.

[10] T. Zimmermann, P. Weisgerber, S. Diehl e A. Zeller, “Mining Version Histories to Guide Software Changes,” em 26th International Conference on Software Engineering, Washington, 2004.

[11] M. D'Ambros, H. Gall, M. Lanza e M. Pinzger, “Analysing Software Repositories to Understand Software Evolution,” em Software Evolution, 1 ed., Berlin, Springer Berlin Heidelberg, 2008, pp. 37-67.

[12] U. M. Fayyad, G. Piatetsky-Shapiro and P. Smyth, "From Data Mining to Knowledge Discovery: An Overview," in Advances in Knowledge Discovery and Data Mining, Menlo Park, CA, USA, American Association for Artificial Intelligence, 1996, pp. 1-34.

[13] S. d. Amo, Técnicas de Mineração de Dados, Uberlândia, Minas Gerais: Universidade Federal de Uberlândia, 2004, p. 43.

[14] G. k. d. Freitas, Um Modelo para Descoberta de Regras de Associação em Repositório de Código Fonte, Maringá: Departamento de Informática, Universidade Estadual de Maringá, 2013.

[15] L. M. R. d. Vasconcelos and C. L. d. Carvalho, "Aplicação de Regras de Associação para Mineração de Dados na Web," Goiânia, 2004.

[16] R. Agrawal, T. Imielinski and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," in ACM SIGMOD Conference, New York, 1993.

[17] K. J. Cios, W. Pedrycz, R. W. Swiniarski and L. Kurgan, Data Mining: A Knowledge Discovery Approach, 1 ed., New York: Springer US, 2007.

[18] C. O. Camilo and J. C. d. Silva, "Mineração de Dados: Conceitos, Tarefas, Métodos e Ferramentas," Goiânia, 2009.

[19] T. J. McCabe, "A Complexity Measure," IEEE Transactions of Software Engineering, v. 2, n. 4, p. 13, 1976.

[20] I. Sommerville, Engenharia de Software, 8 ed., São Paulo: Pearson, 2007.

[21] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Transactions on Software Engineerins, v. 20, n. 6, p. 476-493, Junho 1994.

[22] Q. D. Soetens and S. Demeyer, "Studying the Effect of Refactorings: A Complexity Metrics Perspective," Proceedings of the 2010 Seventh International Conference on the Quality of Information and Communications Technology, pp. 313-318, 2010.

[23] J. L. D. Olmedo e J. M. Vázquez, “Comparación de la Discretización Estándar con un Nuevo Método para Reglas de Asociación Cuantitativas” IEEE América Latina, vol. 14, pp. 1879-1885, 2016.

[24] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Eman, J. Rosenberg. Preliminary guidelines for empirical research in software engineering. Software Engineering, IEEE Transactions on, v. 28, n. 8, p. 721-734, Aug 2002. ISSN ISSN: 0098-5589 DOI: 10.1109/TSE.2002.1027796

[25] S. Easterbrook, J. Singer, M. Storey, D. Damian. Selecting Empirical Methods for Software Engineering Research. In: F. Shull, J. Singer, D. K. Sjoberg. Guide to Advanced Empirical Software Engineering. [S.l.]: Springer London, 2008. p. 285-311. ISBN ISBN: 978-1-84800-043-8 DOI: 10.1007/978-1-84800-044-5_11.

[26] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and I. H. Witten, The WEKA Data Mining Software: An Update, vol. 11, Hamilton: SIGKDD Explorations, 2009.

[27] S. Beecham, T. Hall, C. Britton, M. Cottee and A. Rainer. Using an expert panel to validate a requirements process improvement model, Journal of Systems and Software, Volume 76, Issue 3, June 2005, Pages 251-275, ISSN 0164-1212, http://dx.doi.org/10.1016/j.jss.2004.06.004.