GiveMe Metrics - Um *Framework* Conceitual para Extração de Dados Históricos sobre Evolução de Software

Jacimar F. Tavares[1], Regina Braga[1], José Maria N. David[1], Marco Antônio P. Araújo[2], Fernanda Campos[1]

[1]Programa de Pós Graduação em Ciência da Computação
Universidade Federal de Juiz de Fora (UFJF)
jacimar.tavares@ice.ufjf.br, {regina.braga, jose.david, fernanda.campos}@ufjf.edu.br
[2]Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais - Campus Juiz de Fora (IF Sudeste MG)
marco.araujo@ifsudestemg.edu.br

Resumo. Este artigo apresenta um *framework* conceitual constituído por ferramentas capazes de extrair dados históricos sobre evolução de software de repositórios de código fonte, de defeitos e de processos de desenvolvimento de software. As ferramentas que compõem o *framework* foram obtidas em um processo de mapeamento sistemático. Uma prova de conceito foi realizada onde foram extraídos dados sobre defeitos em projetos do SINAPAD (Sistema Nacional de Processamento de Alto Desempenho) que possui projetos na área da computação. Também foi realizada a análise dos dados extraídos, objetivando responder algumas questões relevantes para o projeto. Ao final, foi possível avaliar a utilização do *framework GiveMe Metrics* (GMM) na extração de dados sobre defeitos.

Palavras-chave: extração dados históricos, repositórios defeitos, prova de conceito, mapeamento sistemático.

1. Introdução

A atividade de desenvolvimento de software, frequentemente, gera um conjunto de artefatos que necessitam ser gerenciados. Esses artefatos são muitas vezes armazenados em repositórios de dados e mantidos ao longo do ciclo de vida do software. Podem fornecer uma gama de informações sobre sua evolução mas, para que isso seja possível, é necessário extrair dados e analisá-los e, em alguns casos, combinar dados provenientes de diferentes repositórios para obter uma determinada informação [Poncin et al., 2011] [Ligu et al., 2013]. Ainda, pode-se extrair dados de um determinado período de tempo, o que pode ser uma tarefa complexa, dependendo de como os dados estejam armazenados [Pedroso, 2013]. Os dados extraídos podem auxiliar organizações de diferentes formas como, por exemplo, através de uma análise que objetiva entender porque os níveis de acoplamento modificam significativamente de uma versão do software para outra.

Este artigo foca na extração de dados de três diferentes tipos de repositórios de dados históricos: (i) de código fonte, (ii) de defeitos e (iii) de processos de desenvolvimento de software. Para que isso seja possível, foi proposto um *framework* conceitual, através da integração de ferramentas existentes, capaz de extrair dados históricos sobre evolução de software. As ferramentas que compõem o *framework GiveMe Metrics* (GMM) foram obtidas em um processo de mapeamento sistemático no qual foram definidos critérios para que uma ferramenta pudesse ser selecionada para compor o *framework*. O mapeamento sistemático completo está disponível em http://implementarsolucoes.com.br/givememetrics.aspx. Com o objetivo de obter indícios sobre a viabilidade de uso do *framework*, uma prova de conceito foi realizada onde foram extraídos dados sobre defeitos em projetos do SINAPAD (Sistema

Nacional de Processamento de Alto Desempenho) (https://www.lncc.br/sinapad/), que realiza pesquisas científicas na área da computação. Também foi realizada a análise dos dados extraídos, objetivando investigar algumas questões importantes para o projeto como, por exemplo, quais projetos mantidos são os maiores alvos de manutenções corretivas. Neste contexto, considerando a Prova de Conceito realizada no SINAPAD, o projeto ***sinapad-framework*** teve a maior porcentagem de defeitos ao longo dos anos e sofreu várias manutenções corretivas, por se tratar de um projeto base para outros. Essa dependência entre projetos faz com que defeitos no ***sinapad-framework*** possam vir a interferir no funcionamento de outros softwares, e ações corretivas foram especificadas no sentido de minimizar esses impactos. Ao final, foi possível obter indícios sobre a viabilidade de utilização do GMM na extração de dados sobre defeitos.

Este artigo está dividido da seguinte forma: a seção 2 apresenta trabalhos relacionados, enquanto a seção 3 apresenta um resumo do mapeamento sistemático realizado para levantamento de ferramentas. A seção 4 apresenta o GMM, e como utilizá-lo para extrair dados históricos de repositórios. A seção 5 apresenta a prova de conceito realizada, através da qual foi possível extrair dados do repositório de defeitos do SINAPAD e realizar a análise dos dados objetivando a descoberta de informações. Por fim, a seção 6 apresenta as considerações finais e os trabalhos futuros.

2. Trabalhos Relacionados

Apresenta-se a seguir alguns trabalhos relacionados com a proposta deste artigo, não sendo objetivo esgotar a literatura sobre o assunto.

FRASR (*Framework for Analyzing Software Repositories*) [Poncin et al., 2011] é um *framework* que permite a análise de processos de desenvolvimento de software armazenados em repositórios de dados. Dentre suas características, destacam-se a capacidade de extrair informações de sistemas de controle de versão e *bug trackers*. Especialmente, é capaz de contabilizar os defeitos encontrados e verificar o estado em que se encontram, como criado, fechado, comentado ou reaberto. No contexto de repositório de defeitos, o GMM também é capaz de extrair dados de *status* de defeitos, como aberto, fechado, comentado ou reaberto, dependendo do tipo de repositório de defeitos que se estiver manipulando.

Em [Christian e Wil, 2006] é apresentado o *ProM Framework*, que atua na extração de dados de *logs* de eventos gerados durante a construção de software, tanto de repositórios (CVS, Subversion) como de ferramentas (*FLOWer*, *WebSphere*, *Staffware*, *PeopleSoft*, *CPN Tools* e *Apache*). Difere do *framework* GMM, que extrai dados de três tipos de repositórios, para obter métricas de código, defeitos e informações sobre processo de desenvolvimento de software.

Em [Burstein et al., 1997] é apresentado o *framework OMIS* que é capaz de lidar com informações e dados de uma organização, visando armazenar, em repositórios de dados, o conhecimento obtido (como modelos conceituais e modelos de processos), criados para representar as práticas organizacionais. A diferença fundamental entre essa proposta e o GMM é que esse último tem por objetivo extrair dados históricos de repositórios, exportando-os em determinados formatos de arquivos. Já o *framework* OMIS tem por objetivo manter os dados extraídos em um repositório, de forma a gerar evidências sobre como a organização evolui suas aplicações.

3. Mapeamento Sistemático

O mapeamento sistemático é um mecanismo que permite ao pesquisador fornecer uma visão geral sobre a área de pesquisa que está sendo investigada, levando em conta fatores como a quantidade de pesquisas publicadas sobre a área pesquisada e os tipos de pesquisa disponíveis [Steinmacher e Gerosa, 2012]. Já um protocolo de mapeamento é o método que o pesquisador define na formalização de buscas sobre a área de pesquisa alvo de investigação [Kitchenham, 2004]. Neste artigo, optou-se pela realização de um mapeamento sistemático para auxiliar na busca por ferramentas, para compor o *framework* GMM*,* dado a formalidade e o rigor que se pode obter ao realizar uma busca.

O protocolo desenvolvido foi constituído de objetivo, critérios para avaliação das ferramentas, questões de pesquisa, *string* de busca e bases de dados de publicações.

Objetivo: levantar quais são as ferramentas capazes de reportar dados sobre evolução de software, provenientes de três categorias de repositórios: Categoria 1 - Repositórios de Código Fonte; Categoria 2 - Repositórios de Defeitos de Software; Categoria 3 - Repositórios de Dados sobre o Processo de Desenvolvimento de Software.

Critérios para a avaliação e aceitação das ferramentas: foram definidos critérios para essa tarefa, estabelecidos com base em uma reunião com pesquisadores, tendo sido apresentadas as características dos repositórios (de código fonte, de defeitos e de processos de desenvolvimento de software), incluindo características utilizadas por outras instituições parceiras, além do SINAPAD. Assim, os critérios foram definidos a partir dos principais critérios levantados na literatura e revisados nessa reunião.

Cada categoria possui um conjunto específico de critérios. Ferramentas da Categoria 1 - Repositórios de Código Fonte, devem possuir as seguintes características:

- (C1): capacidade de se conectar a repositórios de código fonte SVN ou GitHub com suporte a SVN, mesmo sob autenticação;
- (C2): capacidade de analisar código Java ou C#;
- (C3): capacidade de extrair automaticamente um conjunto de métricas por cada *release* ou *commit*, ou por projeto de um software no repositório. Uma ferramenta será considerada válida se extrair pelo menos uma das métricas de cada uma das características de software definidas em [Araújo, 2009], que são Tamanho[1], Periodicidade[2] e Complexidade[3];
- (C4): capacidade de exportar os resultados nos seguintes formatos: .xls, .csv, .txt, .xml ou gravá-los em um banco de dados.

[1] Tamanho: caracterizado pela quantidade de artefatos produzidos em cada etapa do ciclo de vida do software e medido por uma das métricas a seguir: número de pontos de função, números de pontos de caso de uso, número de requisitos, número de classes, número de métodos por classe, número de classes de domínio, número de classes de suporte, número de subsistemas, número de linhas de código fonte.

[2] Periodicidade: representa o intervalo de tempo decorrido entre cada versão produzida de um artefato e medida pelo intervalo de tempo entre versões de um produto (diz respeito à versão do software a partir da qual as métricas são extraídas).

[3] Complexidade: identificada através de elementos que podem medir a complexidade estrutural de um artefato e medida por uma das métricas a seguir: número de casos de uso, número de diagramas de classes, número de diagramas de sequência, número de diagramas de estados, número de diagramas de empacotamento, número de diagramas de atividades, profundidade de herança por classe, número de filhos por classe, acoplamento entre objetos, resposta de uma classe, falta de coesão entre métodos, complexidade ciclomática por método.

Já as ferramentas da Categoria 2 - Repositórios de Defeitos de Software, devem possuir as seguintes características:

- (C1): capacidade de se conectar a repositórios de registro de defeitos, mesmo sob autenticação;
- (C2): capacidade de retornar automaticamente um conjunto de dados históricos por *release, commit* ou por projeto de um software no repositório. Devido à diversidade de tipos de dados que podem ser extraídos sobre um defeito, não será limitado aqui o tipo de defeito registrado. Entretanto, ferramentas que extraiam defeitos de código fonte devem considerar apenas softwares desenvolvidos em Java ou C#;
- (C3): capacidade de exportar os resultados obtidos nos seguintes formatos: .xls, .csv, .txt, .xml, .html ou gravá-los em um banco de dados.

Por fim, as ferramentas na Categoria 3 - Repositórios de Dados sobre o Processo de Desenvolvimento de Software, devem possuir as seguintes características:

- (C1): capacidade de se conectar a repositórios de registro de dados de processos de desenvolvimento de software, mesmo sob autenticação;
- (C2): capacidade de retornar um conjunto de dados históricos por *release, commit* ou projeto de um software no repositório. Uma ferramenta será considerada válida se retornar qualquer tipo de informação sobre o processo de desenvolvimento de software. Uma resalva está no fato de que, caso a ferramenta considere projetos de código para extrair informações, somente serão aceitas ferramentas que manipulem código Java ou C#;
- (C3): Capacidade de exportar os resultados obtidos nos seguintes formatos: .xls, .csv, .txt, .xml, .html ou gravá-los em um banco de dados.

Questões de Pesquisa: mediante o objetivo apresentado e os critérios para avaliação das ferramentas, foram definidas questões de pesquisa, com o objetivo de guiar a execução do mapeamento sistemático. As questões de pesquisa definidas são as seguintes:

- (Q1) Quais são as ferramentas disponíveis em cada uma das categorias de repositórios?
- (Q2) Em quais critérios pré-definidos para cada uma das categorias de repositórios as ferramentas se encaixam?

Strings de busca: para cada uma das categorias de repositórios, foram definidas as seguintes *strings* de busca:

- Categoria 1**:** ((ferramenta OR tool OR plugin) AND (extração OR extract OR mining OR mineração) AND (métricas OR métrica OR metric OR metrics) AND (repositório OR repositórios OR repository OR repositories) AND ((código AND fonte) OR (source AND code)))
- Categoria 2: ((ferramenta OR tool OR plugin) AND (extração OR extract OR mining OR mineração) AND (métricas OR métrica OR metric OR metrics) AND (repositório OR repositórios OR repository OR repositories) AND (defeitos OR defeito OR defect OR defects OR bug OR bugs OR "bug tracker"))
- Categoria 3:((ferramenta OR tool OR plugin) AND (extração OR extract OR mining OR mineração) AND (métricas OR métrica OR metric OR metrics) AND (repositório OR repositórios OR repository OR repositories) AND (software AND development AND process))

Bases de dados de publicações: para executar as *strings* de busca, foram consideradas cinco bases de dados eletrônicas de publicações com acesso livre para alunos da Universidade Federal de Juiz de Fora (UFJF). As bases usadas foram:

- Science@Direct (http://www.sciencedirect.com)
- El Compendex (http://www.engineeringvillage.com)
- IEEE Digital Library (http://ieeexplore.ieee.org)
- ISI Web of Science (www.isiknowledge.com)
- Scopus (http://www.scopus.com)

O protocolo do mapeamento sistemático definido nesta seção permite uma busca formal pelas ferramentas, diferentemente de uma abordagem de pesquisa *ad-hoc* que não possui uma metodologia rigorosa para obter resultados através de investigação [Kitchenham, 2004].

O mapeamento foi realizado no mês de novembro de 2013. Os resultados obtidos mostram que foi possível recuperar ferramentas proprietárias e livres, sendo que na Categoria 1 foram encontradas 12 ferramentas, sendo que destas, apenas 2 contemplaram todos os critérios definidos no protocolo (*Analizo* e *Kalibro Desktop*). Na Categoria 2 foram encontradas 22 ferramentas, onde apenas 7 obedeceram aos critérios definidos (*BuCo Reporter*, *Bugzilla*, *Bug Track*, *Redmine*, *Mantis*, *Hudson* e *Jenkins*). Já na Categoria 3, foram encontradas 5 ferramentas, onde apenas 3 foram consideradas (*Issue Player*, *DIG* e *Disco*). A lista completa de ferramentas recuperadas, bem como demais resultados obtidos, estão disponíveis em http://implementarsolucoes.com.br/givememetrics.aspx.

4. Um *Framework* Conceitual para Extração de Dados Históricos sobre Evolução de Software

A proposta do GiveMe Metrics *framework* surgiu da necessidade de extrair dados históricos de três diferentes tipos de repositórios, seguindo critérios específicos (aqueles descritos no protocolo do mapeamento). A maior vantagem/contribuição na sua utilização está nos parâmetros fornecidos para a escolha de ferramentas, dadas as características do repositório que se deseja manipular, visando simplificar a escolha da que melhor se encaixa nas necessidades da tarefa de manipulação do repositório.

Foram levados em consideração os tipos de repositórios de dados importantes para a extração de dados históricos sobre o ciclo de vida do software (repositórios mencionados no protocolo) e os grupos de ferramentas identificados no mapeamento. É importante destacar que todas as ferramentas que se encaixam nos critérios descritos no protocolo foram usadas na concepção do *framework* onde, em um total de 39 ferramentas avaliadas, apenas 12 foram selecionadas. Algumas delas possuem limitações, como falta de documentação e clareza na definição de como as métricas são obtidas, mas por se encaixarem em todos os critérios, foram consideradas. Tais ferramentas serão apresentadas mais adiante. A Figura 1 apresenta uma visão de alto nível dos principais fluxos do *framework* GMM.

A utilização do *framework* pode ser dividida em três diferentes cenários, permitindo ao usuário escolher entre três diferentes tipos de repositórios de dados para analisar: repositório de código fonte, repositório de defeitos ou repositório de dados sobre o processo de desenvolvimento de software.

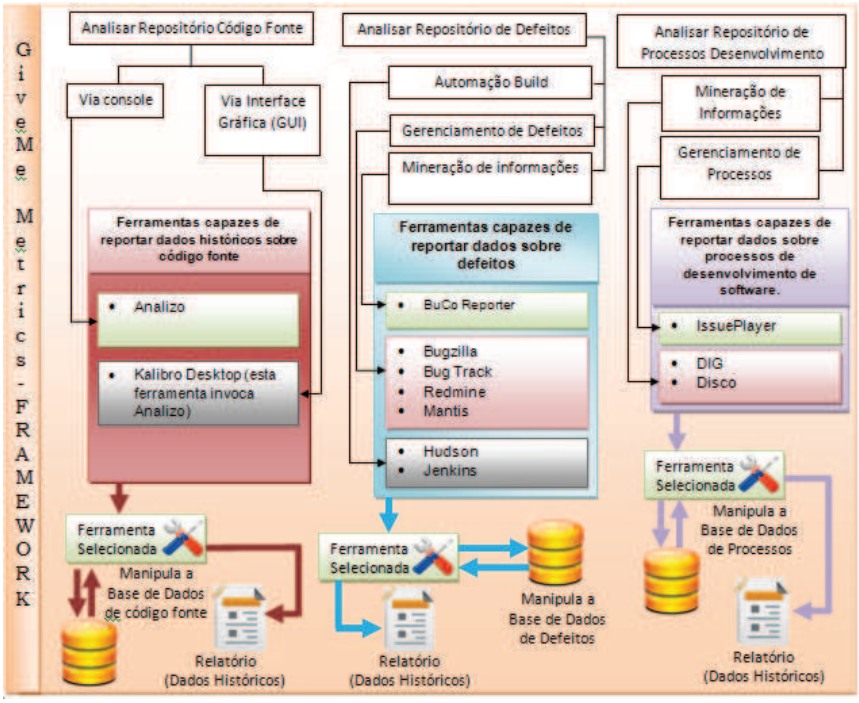

Figura 1. *GiveMe Metrics Framework*

Cenário 1 - Analisar Repositório de Código Fonte: o usuário poderá escolher entre uma ferramenta manipulável via console (através de linha de comando) ou uma ferramenta manipulável via interface gráfica do usuário (GUI). Ambas as ferramentas (*Analizo* ou *Kalibro Desktop*) permitem a extração de métricas de código Java e são baseadas em software livre. Entre as métricas definidas no critério C1 da Categoria 1 do protocolo de mapeamento, *Analizo* e *Kalibro Desktop* são capazes de extrair as seguintes métricas em cada uma das características de software:

- Tamanho: número de classes, número de métodos por classe, número de subsistemas, número de linhas de código fonte;
- Periodicidade: intervalo de tempo por versão de um projeto;
- Complexidade: profundidade de herança por classe, número de filhos por classe, acoplamento entre objetos, falta de coesão entre métodos e complexidade ciclomática.

Após a seleção da ferramenta, pode-se manipular a base de dados de código fonte, que significa executar a ferramenta sobre as versões de um projeto que estão sendo versionadas no repositório, e obter determinadas métricas sobre cada uma das versões. Ao final, obtém-se um conjunto de dados sobre o software, nesse caso, métricas de código fonte.

Cenário 2 - Análise de Repositórios de Defeitos: o mapeamento sistemático realizado anteriormente permitiu a descoberta de dois formatos diferentes de registro de defeitos: (i) registros textuais (ou *Tickets*), manualmente cadastrados sobre um defeito encontrado no software, ou (ii) um defeito automaticamente detectado (com uso de ferramentas de verificação de código fonte) ao analisar o código fonte de um software, como problemas no uso dos recursos da linguagem de programação, problemas na API de desenvolvimento e exceções que não foram tratadas. A extração de dados sobre defeitos pode ser feita usando três diferentes grupos de ferramentas, como pode ser visto

na Figura 1: ferramentas de automação de *build*, ferramentas de gerenciamento de defeitos e ferramentas de mineração de informações em repositórios do tipo *Bugzilla*. Caso os defeitos cadastrados sejam normalmente acessados pelas ferramentas *Hudson* ou *Jenkins* (ambas baseadas em software livre), as mesmas poderão ser usadas para extrair dados sobre os defeitos. Se a organização utilizar ferramentas como *Bugzilla* (software livre e *open source*)*, Bug Track* (software livre)*, Redmine* (software livre) ou *Mantis* (software livre e *open source*) para gerenciamento de defeitos encontrados em um projeto de software, poderão ser usadas também para extrair dados sobre defeitos. Um terceiro cenário é o de mineração de informações em repositórios como *Bugzilla*, possível com o uso da ferramenta *BuCo Reporter* (software livre). Após a escolha da ferramenta, a base de dados de defeitos é minerada e dados extraídos dos defeitos são obtidos.

Com as ferramentas *Hudson* e *Jenkins* é possível extrair, através de suas funcionalidades de exportação, dados provenientes de bases de dados de defeitos como o *Bugzilla, Redmine* e *Mantis*. Isso é possível através da instalação de *plugins* compatíveis. O tipo de dado que pode ser extraído com essas ferramentas depende do tipo de *plugin* e da base de dados de defeitos integrada ao *Hudson* ou *Jenkins*. Em [Hudson, 2013] e [Jenkins, 2013] é possível encontrar informações sobre esses *plugins*.

Com as ferramentas de gerenciamento de defeitos *Bugzilla, Bug Track, Redmine* ou *Mantis* é possível extrair dados como defeitos em aberto, defeitos resolvidos, autor dos defeitos, descrição do defeito, dentre outros.

Com a ferramenta *BuCo Reporter* é possível minerar informações como número de defeitos não resolvidos, número de defeitos resolvidos, tempo médio de correção de um defeito, idade média de um defeito não resolvido, número de defeitos detectados depois do lançamento de uma *release*, relação entre defeitos resolvidos e não resolvidos, número de defeitos detectados depois de um período de tempo determinado e porcentagem de defeitos documentados.

Cenário 3 - Analisar Repositórios de Processos de Desenvolvimento: o usuário poderá escolher entre o grupo de ferramentas de mineração de informações ou o grupo de gerenciamento de processos. No primeiro grupo têm-se as ferramentas *DIG* e *Disco*. No segundo grupo tem-se a ferramenta *Issue Player*.

Com a ferramenta *DIG* é possível obter dados sobre o processo de desenvolvimento de software, bem como quais são as tarefas concluídas e as não concluídas, além do quanto falta para que sejam finalizadas. Há também a coleta de medidas de esforço por membro da equipe no projeto. Entretanto, a documentação disponível para essa ferramenta não é clara em expor como tais dados são calculados. Há apenas a citação textual de que são coletados, mas não há detalhes, por exemplo, de como é medido o esforço de um membro na equipe. Com a ferramenta *Disco* tem-se a mesma situação, a falta de clareza ao expor como os dados são obtidos. *Disco*, em seu *site*, apresenta uma lista de dados sobre o processo de desenvolvimento que podem ser obtidos, mas não apresenta uma descrição do que significam. Apesar disso, tais ferramentas não foram excluídas do *framework*, pois se encaixam em todos os critérios pré-estabelecidos no mapeamento sistemático. Os dados possíveis de se obter são, entre outros: frequência absoluta, número máximo de repetições, duração total (não está explícito se considera duração de uma tarefa ou projeto) e a duração máxima. Já com a ferramenta *Issue Player* é possível extrair dados como média de tempo dos artefatos no

repositório, média de tempo dos defeitos cadastrados pertencentes a um projeto, média de tempo dos arquivos no repositório, média de tempo de construção das funcionalidades de um software.

Além da extração dos dados propriamente ditos, pode-se ainda trabalhar aspectos de visualização de informações, selecionando estratégias para visualizar os resultados obtidos na etapa de extração. Exemplos de estratégias podem ser vistos em [Ribeiro, 2013], onde são apresentadas heurísticas utilizadas no cenário de dados governamentais para verificar se as visualizações escolhidas são as mais adequadas aos dados que estão sendo analisados no momento.

5. Prova de Conceito

Com o intuito de verificar a viabilidade de utilização do *GiveMe Metrics*, uma Prova de Conceito (*Proof of Concept - PoC*) foi executada. Essa Prova de Conceito foi criada em um cenário real, que é o do SINAPAD. O objetivo desta PoC foi definido de acordo com a abordagem *Goal/Question/Metric* (GQM) [Basili et al., 1994]. Os objetivos, segundo essa abordagem, devem ser formulados conforme o *template* "Analisar o <*objeto de estudo*> com a finalidade de <*objetivo*> com respeito a <*foco da qualidade*> do ponto de vista de <*perspectiva*> no contexto de <*contexto*>". Com base no *template* para objetivos do GQM, o objetivo desta prova de conceito é: "Analisar o *framework GiveMe Metrics* com a finalidade de verificar a viabilidade para a análise de projetos de software no contexto de projetos em computação científica".

O cenário analisado compreende o repositório de defeitos do SINAPAD, usado para registro dos defeitos detectados pela equipe de desenvolvimento e suporte. Dentre os projetos do SINAPAD, podem-se destacar os projetos dos portais SINAPAD, que são portais científicos que permitem a execução, via *interface* Web, de aplicações científicas diversas, nos *clusters* dos centros que compõem o sistema nacional e o ***sinapad-framework***, que é um *framework* que provê recursos para outros projetos. Conforme dito anteriormente, o SINAPAD é uma rede de centros de computação de alto desempenho, geograficamente distribuídos, instituída pelo Ministério da Ciência e Tecnologia do Brasil. A extração dos dados foi autorizada pelo SINAPAD e acompanhada pelo responsável pelo repositório de defeitos.

5.1 Definição da ferramenta para extração dos dados

O primeiro passo para utilizar o *framework* GMM neste cenário foi a escolha do tipo de repositório que se deseja extrair dados. No caso desta *PoC*, o repositório escolhido foi o repositório de defeitos do SINAPAD.

O segundo passo é a escolha da ferramenta a ser utilizada para extração dos dados. Para defini-la, foi analisado como o SINAPAD mantém o repositório de defeitos. Pode-se verificar que os defeitos detectados pelos pesquisadores são cadastrados em forma de *tickets* em um sistema gerenciador de defeitos, o *Redmine* [Redmine, 2013]. Como há a utilização de um sistema de gerenciamento de defeitos, foi analisado o grupo Gerenciamento de Defeitos, pertencente à categoria Repositório de Defeitos (ver Figura 1), objetivando verificar se a *Redmine* é uma das ferramentas disponíveis no GMM. Como o GMM engloba a ferramenta *Readmine*, foi possível utilizar a própria ferramenta de gerenciamento de defeitos que o SINAPAD utiliza.

O terceiro passo é a utilização dos mecanismos de exportação do *Redmine* para extrair os dados sobre defeitos do repositório de defeitos, que corresponde à etapa de Manipular a Base de Dados de Defeitos do GMM (ver Figura 1). No caso do SINAPAD, o repositório de defeitos é um banco de dados relacional (como *MySQL* e *SQL Server*, por exemplo), responsável por armazenar todos os registros de defeitos detectados durante a evolução dos seus projetos de software.

5.2 Definição dos dados a serem extraídos

A *Redmine* permite a extração de dados e exportação em formato CSV, havendo a possibilidade de escolha do tipo de dado a ser extraído como, por exemplo, data de cadastramento do defeito, para quem foi atribuído, dentre outros. Nesta *PoC* deseja-se investigar algumas questões, em específico (listadas a seguir), objetivando entender quais são os projetos que o SINAPAD tem trabalhado atualmente na tentativa de identificar os projetos que mais sofreram manutenção corretiva, isto é, aqueles que mais apresentaram defeitos ao longo dos anos. Isso permitirá ao SINAPAD identificar quais projetos têm consumido mais tempo da equipe responsável por realizar manutenções corretivas e direcionar uma equipe que possa melhorar o projeto de código do referido sistema, na tentativa de reduzir a porcentagem de novos defeitos no futuro no projeto em questão:

- (P1) Quais são os projetos alvos de defeitos?
- (P2) Qual a porcentagem de defeitos cadastrados para cada um dos projetos?
- (P3) Qual a porcentagem de defeitos com prioridade alta de correção?
- (P4) Como os defeitos estão divididos ao longo dos anos?

Para responder a essas perguntas, foram extraídos os seguintes dados: identificador do defeito, situação do defeito (aberto, fechado ou em andamento), projeto a qual o defeito pertence, tipo (serve para identificar se trata realmente de um defeito identificado no projeto – apenas o tipo defeito será considerado), prioridade, título do defeito, versão do projeto, início (data de cadastramento do defeito) e criado em (data de cadastramento do defeito). Por questões de confidencialidade, os dados não poderão ser aqui disponibilizados.

5.3 Análise dos Dados

Esta etapa visa a análise dos dados extraídos, objetivando apenas a seleção dos que são importantes para responder às perguntas de pesquisa anteriormente apresentadas. Para transformá-los em informações foi realizada a tarefa de eliminação dos registros contidos no arquivo exportado pela ferramenta *Redmine* que possui tipo diferente de "Defeito". Isso foi necessário porque a base de defeitos também é utilizada para armazenar registros de solicitação de novas funcionalidades. Somente os dados cadastrados como "Defeitos" são de fato considerados defeitos no sistema, o restante refere-se a manutenções adaptativas e evolutivas. É importante destacar que as informações resultantes dessa eliminação foram validadas junto ao Administrador do repositório de defeitos do SINAPAD.

Ao analisar os dados extraídos, foram identificados os projetos para os quais há registro de defeitos, objetivando responder à pergunta P1. A lista completa, e a descrição de cada um delas está disponível em

http://implementarsolucoes.com.br/givememetrics.aspx. Para responder à pergunta P2, foram analisados os defeitos cadastrados para cada um dos projetos. A Figura 2 apresenta os dados relativos à porcentagem de defeitos por projetos, prioridade dos defeitos e distribuição de defeitos por ano.

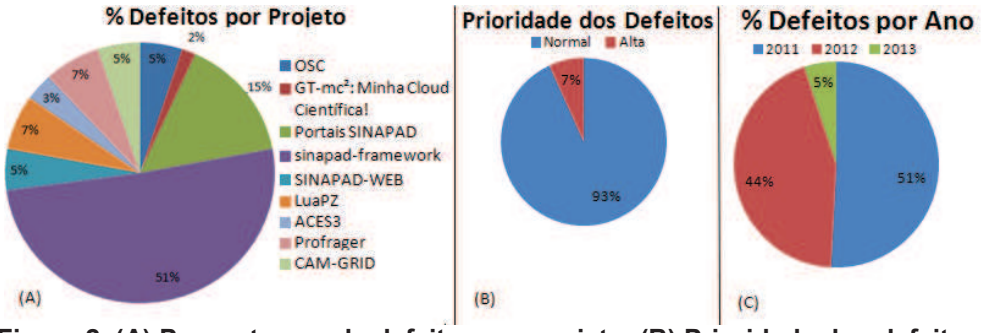

Figura 2. (A) Porcentagem de defeitos por projeto, (B) Prioridade dos defeitos e (C) Defeitos por ano em um projeto

Ao analisar o gráfico da Figura 2(A) é possível perceber que a maior parte dos defeitos (51%) pertence a um único projeto. Segundo o administrador do sistema, isso se dá porque o projeto ***sinapad-framework*** é o maior projeto do SINAPAD, juntamente com os Portais SINAPAD, e é a base para outros projetos, fornecendo, por exemplo, bibliotecas de uso comum.

Para responder à pergunta P3, foi gerado o gráfico da Figura 2(B). Nele é possível perceber que apenas 7% dos defeitos tem prioridade alta. Segundo o administrador, isso se dá porque todos os defeitos são cadastrados como normais, e só há cadastramento de defeitos com prioridade alta quando o nível de criticidade do defeito encontrado é alto a ponto de interromper o uso do sistema em que ele foi detectado. Defeitos com prioridade alta devem ser imediatamente resolvidos.

Para responder à pergunta P4 foram utilizados os dados de criação de um defeito. Todas as datas de lançamento de novos defeitos foram analisadas e agrupadas por ano. Verificou-se que os defeitos estão divididos entre os anos de 2011, 2012 e 2013. Analisando o gráfico da Figura 2(C), é possível perceber que o ano com maior número de defeitos cadastrados é o ano de 2011, com 51% dos defeitos, seguido do ano de 2012, com 44%, e 2013, com 5%. Segundo o administrador, isso se deu porque 2011 foi o ano de desenvolvimento do maior projeto do SINAPAD, o ***sinapad-framework***. Isso, na prática, fez com que a cada etapa do desenvolvimento, teste e utilização do projeto, novos defeitos fossem detectados e cadastrados.

Ao final, têm-se as perguntas respondidas com base na análise dos dados extraídos. Foi possível perceber que o rigor na coleta dos dados pode variar entre instituições, pois os dados sobre um defeito armazenado dependem dos objetivos da organização e do nível de controle desejado sobre os defeitos em seus projetos.

Uma das mais importantes contribuições das análises realizadas nos dados extraídos foi a identificação de que o ***sinapad-framework*** é o projeto com maior ocorrência de defeitos ao longo dos anos, e que defeitos nesse projeto podem acarretar problemas em outros projetos que se baseiam nele. Assim, uma ação possível é direcionar uma equipe de manutenção para evoluir o projeto de código do ***sinapad-framework***, melhorando-o e realizando, caso ainda não seja feito, testes unitários e

funcionais, diminuindo a incidência de novos defeitos, o que consequentemente poderá reduzir o número de defeitos em outros projetos que dependem dele.

Neste trabalho foram apresentadas ferramentas com o propósito de extrair diversos tipos de dados, analisados separadamente. Um cenário que não foi explorado envolve a possibilidade de armazenar dados relevantes gerados a partir de experimentações realizadas. Para que isso seja possível, é necessário certificar-se da corretude do processo de extração de dados, e rastrear sua origem. Em [Melo et al., 2011] é dito que a proveniência de dados pode ser útil na gerência de dados obtidos nesses estudos experimentais

6. Considerações finais e trabalhos futuros

Este artigo apresentou o *framework GiveMe Metrics*, capaz de extrair dados de três diferentes tipos de repositórios de dados históricos. Evidenciou-se que, entre os projetos mantidos pelo SINAPAD, o ***sinapad-framework*** é o que apresentou maior porcentagem de defeitos ao longo dos anos. Uma ação possível é evoluir o projeto de código do ***sinapad-framework*** objetivando reduzir a incidência de novos defeitos, aliada à aplicação de técnicas e ferramentas de controle da qualidade de software.

Mediante aos resultados obtidos, uma hipótese de pesquisa foi formulada, considerando avaliar se o aumento da porcentagem de defeitos apresentada em um projeto influencia no aumento do tempo médio gasto, em dias, para resolver problemas neste mesmo projeto:

- H $_{nula}$: não existe correlação forte entre a porcentagem de defeitos de cada um dos projetos com a média de tempo gasto em dias na resolução de defeitos;
- H $_{alternativa}$: existe correlação forte entre a porcentagem de defeitos de cada um dos projetos e a média de tempo gasto, em dias, na resolução de defeitos em um mesmo projeto.

Dada a hipótese levantada, foi aplicado o Coeficiente da Correlação de Pearson [Levin et al., 2012] visando descobrir, além da direção da correlação, também se a correlação é nula, fraca, moderada, forte, ou perfeita. Todo o processo de cálculo do Coeficiente de Pearson está disponível em http://implementarsolucoes.com.br/givememetrics.aspx. Os resultados obtidos mostram que, a porcentagem de defeitos por projeto se relaciona fracamente com o tempo médio gasto na resolução de um defeito, permitindo assim que a Hipótese nula seja aceita e a Hipótese alternativa seja rejeitada. Na prática, isso significa que, não necessariamente, sempre que a porcentagem de defeitos por projeto aumenta, o tempo médio gasto para resolução de um defeito também aumentaria na mesma proporção.

As limitações do GiveMe Metrics estão ligadas às ferramentas que podem ser utilizadas na extração de dados históricos. Algumas ferramentas possuem documentação limitada dos recursos oferecidos, bem como das métricas que podem ser obtidas.

A prova de conceito apresentada possui a limitação de abranger apenas a extração de dados de defeitos, não contemplando a extração de dados de código fonte e de processos de desenvolvimento, o que seria interessante para provar a sua utilidade em todos os cenários abrangidos pelo *framework*.

Como trabalho futuro, espera-se captar novos parceiros que possuam repositórios de código fonte, de defeitos e de processos de desenvolvimento de software para que seja possível a extração de dados pelo *framework* GMM e, com isso, ampliar a avaliação dessa proposta. Futuramente, uma base de dados, contendo métricas provenientes dos dados extraídos com o *framework* GMM será criada, onde o conhecimento obtido através da análise dos dados poderá ser armazenado e mantido para consulta por pesquisadores, profissionais da indústria ou através de ferramentas que necessitem acessar as informações contidas nessa base de dados.

Referências

Araújo, M.A.P., (2009), Um Modelo para Observação de Evolução de Software. Tese de Doutorado. COPPE/UFRJ.

Basili, Victor R. Caldiera, Gianluigi H. Rombach, Dieter. (1994). The Goal Question Metric Approach. Chapter in Encyclopedia of Software Engineering, Wiley, 1994.

Burstein, F.; Linger, H.; Zaslavsky, A.; Crofts, N., (1997). "Towards an information systems framework for dynamic organisational memory," *System Sciences, Proceedings of the Thirtieth Hawaii International Conference on* , vol.2, no., pp.262,270.

Christian W. Günther and Wil M. P. van der Aalst. (2006). A generic import framework for process event logs. In *Proceedings of the 2006 international conference on Business Process Management Workshops*, Johann Eder and Schahram Dustdar (Eds.). Springer-Verlag.

Hudson Plugins: http://hudson-ci.org/PluginCentral3/. Acessado em: 26 de novembro de 2013.

Jenkins Plugins: https://wiki.jenkins-ci.org/display/JENKINS/Plugins. Acessado em: 26 de novembro de 2013.

Kitchenham, Barbara. Procedures for performing systematic reviews. (2004) Technical Report, TR/SE-0401, Department of Computer Science, Keele University, Keele,UK.

Levin, Jack. Fox, James Alan. Forde, David R. Estatística para Ciências Humanas. (2012). 11ª ed. São Paulo: Pearson Education do Brasil.

Ligu, Elvis. Chaikalis, Theodoros. Chatzigeorgiou, Alexander. (2013). BuCo Reporter: Mining Software and Bug Repositories.

Melo, Cláudia. Viana, Sidney. Pavón, Judith. Almeida jr, Jorge. (2011) Proveniência de dados em experimentos de software: uma proposta de melhoria sob a ótica da qualidade de dados. Revista de Sistemas e Computação, Salvador, v. 1, n. 1, p. 48-63, jan./jun (2011).

Pedroso, Louise. Revoredo, Kate. Baião, Fernanda (2013). Uma Abordagem Baseada em Mineração de Dados para Apoio ao Ciclo de Vida de Projetos de Pesquisa e Inovação. IX Simpósio Brasileiro de Sistemas de Informação. 2013. João Pessoa, PB.

Poncin, W. Serebrenik, A. Van den Brand, M., (2011) "Process Mining Software Repositories," (*2011).Software Maintenance and Reengineering (CSMR), 15th European Conference on* , vol., no., pp.5,14, 1-4.

Redmine: http://www.redmine.org/. Acessado em: 26 novembro de 2013.

Ribeiro, Fernanda. Caetano Bárbara. Paula, Melise, Chaves, Miriam. Silva, Vinícius. Rodrigues, Sergio Assis. Souza, Jano M. (2013). Heurísticas para Visualização de Dados. IX Simpósio Brasileiro de Sistemas de Informação. 2013. João Pessoa, PB.

Steinmacher, I. Chaves, A.P. Gerosa, M.A. (2012). Awareness Support in Distributed Software Development: A Systematic Review and Mapping of the Literature. Journal of Computer Supported Cooperative Work.