BugInf: API Livre para suporte de Visualização de Repositórios de *bug tracking* usando *ElasticSearch* e *Kibana*

Diego J. Oschoski[1], **Guilherme S. de Lacerda**[1]

[1]Faculdade de Informática
Centro Universitário Ritter Dos Reis (UNIRITTER)
Porto Alegre – RS – Brazil

{silvaoschoski,guilhermeslacerda}@gmail.com

Resumo. *É muito comum o uso de sistemas de bug tracking para gerenciamento de defeitos por times de software. No entanto, o formato que as informações são exibidas por estas ferramentas são de difícil interpretação e visualização. Este trabalho apresenta o BugInf, API livre para apoiar a coleta de informações de sistemas de bug tracking e posterior visualização destas, apoiando times de software nas tomadas de decisões. Estas visualizações são possíveis com o uso de outras duas ferramentas livres: ElasticSearch e Kibana. Foi realizado um experimento inicial com a API usando o repositório do Redmine, em que foi possível analisar estas informações de forma visual e consolidada, apresentando um grande potencial para futuros trabalhos.*

Abstract. *It is very common to use bug tracking systems for defect management for software teams. However, the format information is displayed by these tools are difficult to interpret and display. This work presents the BugInf, free API to support the collection information from bug tracking systems information to visualize them, supporting software teams in decision making. These views are possible with the use of two other free tools: ElasticSearch and Kibana. We conducted an initial experiment with the API using the Redmine repository, it was possible to analyze this information in visual and consolidated form, with great potential for future work.*

1. Metadata

Version described in this paper: 1.0

License: LGPL 2.1

Link to source code repository: https://github.com/diegojoseoschoski/buginf

Link to project website: : https://github.com/diegojoseoschoski/buginf

Link to communication channel: {silvaoschoski,guilhermeslacerda}@gmail.com

Programming languages used: Java

Dependencies: Java 7, Maven 3, ElasticSearch 1.5.2, Kibana 4.0.2

Operating Systems Compatible: All (*Java Virtual Machine* support)

List of contributors: Diego Oschoski, Guilherme Lacerda

2. Introdução

Devido a complexidade e tamanho dos projetos de software, é comum que apareçam diversos defeitos durante o seu ciclo de desenvolvimento e manutenção [Dal Sasso and Lanza 2014, Lata and Sharma 2015]. Para se ter um maior controle destes defeitos, os times usam sistemas de *bug tracking* [Mockus et al. 2002, Bertram 2009]. Os principais usuários destes sistemas são desenvolvedores, testadores e times de qualidade, no qual sua utilidade é fornecer feedback sobre os *bugs* ou para corrigí-los.

Os dados armazenados nos repositórios destes sistemas de *bug tracking* podem gerar diversas informações úteis dos softwares, como por exemplo o acompanhamento da evolução da qualidade ao longo do tempo e identificação dos componentes dos sistemas com a maior incidência de defeitos [D'Ambros et al. 2007]. No entanto, o formato que as informações são exibidas nos sistemas de *bug tracking* é de difícil interpretação quando se trata de visualizá-las. Neste contexto, a visualização destas informações são relevantes, permitindo uma análise em diversos aspectos dos softwares.

Na literatura, existem diversas pesquisas abordando a análise de repositórios de sistemas de *bug tracking*. O objetivo é identificar a duplicidade de defeitos [Wang et al. 2008], predição de defeitos [D'Ambros et al. 2012] e evolução dos softwares [D'Ambros and Lanza 2006]. Estes trabalhos visam a simplificar as atividades dos envolvidos nos projetos no sentido da manipulação dos defeitos como parte do processo de desenvolvimento e também compreender quais partes dos sistemas são mais problemáticas.

No contexto abordado anteriormente, existem alguns trabalhos que abordam a análise de repositórios de sistemas de *bug tracking*. Quando o foco é ferramentas que possibilitem a análise visual de repositórios de diferentes sistemas de *bug tracking*, a literatura apresenta um trabalho que propõe uma ferramenta que possibilita sua extensão para outros repositórios de sistemas de *bug tracking* [Dal Sasso and Lanza 2013, Dal Sasso and Lanza 2014].

Neste trabalho, é proposta uma API[1] livre chamada *BugInf*. Ela provê mecanismos para automatizar a coleta e disponibilização das informações, usando duas ferramentas de sofware livre para realização de análises visuais dos dados armazenados nos repositórios de sistemas de *bug tracking*. O propósito é fornecer visões personalizadas com a finalidade de apoiar o acompanhamento da evolução da qualidade do software e ajudar na tomada de decisão dos gestores e da equipe em relação a qualidade do produto.

O trabalho esta organizado da seguinte forma: na seção 3, é introduzido o conceito de visualização de informações e sistemas de *bug tracking*, bem como o entendimento do processo de *bug tracking* e principais ferramentas. Os trabalhos relacionados no contexto de análise visual de informações armazenadas em sistemas de *bug tracking* são discutidos na seção 4. A API livre *BugInf*, bem como sua arquitetura, objetivos, propósito e seus componentes são discutidas na seção 5. Ainda na seção 5, é apresentado os experimentos preliminares realizados e resultados obtidos até o momento. Finalmente, na seção 6 é discutida as conclusões parciais e trabalhos futuros.

[1] *Application Programming Interface*

3. Fundamentação

3.1. Visualização de Informações

A visualização de informações é uma área de pesquisa da computação gráfica que procura criar e desenvolver técnicas de visualização para grandes quantidade de informações [Jean-Daniel Fekete 2008]. Estas informações podem vir de diversas fontes de dados, tais como páginas *web*, estrutura de diretórios e arquivos, além de outros tipos de dados abstratos [Gershon and Eick 1997].

Esta necessidade de novas formas que facilitem a representação das informações decorre da necessidade de como lidar com várias fontes vindas de diferentes aplicações. A visualização de informações tem como objetivo principal facilitar o entendimento por intermédio de representações visuais que tendem a proporcionar uma forma mais simples e intuitiva para entender melhor e mais rapidamente o significado dos dados representados [Gershon and Eick 1997].

Para utilizar a visualização de informações no apoio à resolução de problemas ou na análise de dados, é necessário o conhecimento do sistema no contexto em que será aplicado, assim como das necessidades de visualização e agregação das funcionalidades com o objetivo de tornar as visualizações efetivas. Torna-se necessário, primeiramente, estudar o problema e posteriormente verificar a necessidade de utilizar técnicas de visualizações.

É relevante também entender melhor como o ser humano interage com a informação, ou seja, como ele a percebe visualmente e não visualmente, bem como a mente trabalha quando está procurando por informações conhecidas e não conhecidas [Cemin 2001]. A Figura 1 ilustra, de uma maneira resumida, um processo automatizado de visualização de informações, em que uma imagem é produzida, a partir de dados de entrada.

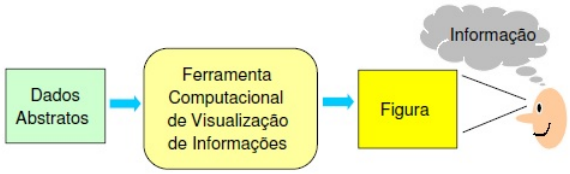

Figura 1. Processo simplificado de visualização de informações auxiliado por computador [Do Nascimento and Ferreira 2005]

3.2. Sistemas de *Bug Tracking*

Bug tracking é o processo de elaboração de relatórios e acompanhamento do progresso de defeitos, desde a sua descoberta até a sua resolução. Um defeito é definido como um desvio de requisitos [Kumar et al. 2013]. Outras terminologias frequentemente utilizadas para descrever este processo inclui gestão da mudança, falhas e incidentes.

Sistemas de *bug tracking* são mais frequentemente usados nas fases de codificação e testes no processo de desenvolvimento de software. No entanto, estes sistemas podem, na verdade, ser utilizados para muitos outros fins, tais como rastreamento de questões em geral, listas de tarefas, *help desk*, gerenciamento de contatos, solicitações de melhorias, entre outras funcionalidades [Kumar et al. 2013].

Diversos projetos fazem uso de sistemas de *bug tracking*. Em sua maioria, estão os projetos de sofware livre que possuem uma forte adoção desses sistemas [Mockus et al. 2002, Zimmermann et al. 2009]. Se bem utilizados, os sistemas de *bug tracking* podem trazer diversos benefícios [Janák 2009], tais como melhoria na qualidade de software, aumento da satisfação dos usuários e clientes, aumento da produtividade da equipe e redução de despesas de projeto, entre outros.

3.3. Ferramentas de *Bug Tracking*

Atualmente, existem diversas opções de ferramentas de *bug tracking*. Estas, em sua maioria, são sofwares livres, possibilitando sua adoção sem custos pelas empresas ou projetos. Conforme [Ghahrai 2015], é apresentado uma lista das principais ferramentas livres (tabela 1).

Ferramenta	Descrição
Redmine	Desenvolvido em Ruby on Rails, Redmine é uma aplicação web de gerenciamento de projetos flexível, multiplataforma e que possui suporte à múltiplos bancos de dados. Gráficos de *Gantt, wiki* do projeto, controle de tempo, controle de acesso baseado em papéis, campos personalizados, vários idiomas, notícias, documentos e gerenciamento de arquivos são algumas das principais características desta ferramenta [Redmine 2016].
Bugzilla	Bugzilla é uma ferramenta que auxilia na gestão de desenvolvimento de software. Desenvolvido pela *Mozilla Foundation*, é uma ferramenta que permite gerenciar o processo de desenvolvimento, manter o controle de bugs, bem como o rastreamento das alterações de código [Bugzilla 2016].
Mantis BT	Este é um dos sistemas de rastreamento de defeitos bem conhecido de sofware livre desenvolvido em *PHP*. Ele suporta diferentes bancos de dados, incluindo *MS SQL, MySQL, PostgreSQL*, entre outros. É um sistema de rastreamento de *bugs* simples e fácil de usar. A integração do código-fonte, controle de tempo, campos personalizados e fluxos de trabalho e outras funcionalidades são algumas das principais características deste sistema [BT 2016].
Trac	É um projeto web, sofware livre, que é escrito em *Python*. Com a sua abordagem minimalista para a gestão, relatórios, funções de rastreamento, gerenciamento de usuários e suporte para vários *plugins*. Além disso, faz rastreamento de *bugs* e gestão para os desenvolvedores. Ela suporta, ainda, múltiplas plataformas como *Mac OS X, Windows, Linux e UNIX* [Trac 2016].

Tabela 1. Descrição das principais ferramentas livres de *bug tracking*

Na próxima seção, são discutidos alguns trabalhos que abordam ferramentas que

apoiam a visualização de informações de *bug tracking*.

4. Trabalhos Relacionados

D'Ambros e outros [D'Ambros et al. 2007] utilizaram um repositório de um sistema de *bug tracking* com dados históricos de defeitos cadastrados para propor uma abordagem para visualizar o ciclo de vida dos defeitos, chamados *System Radiography* e *BugWatch*. Estas técnicas de visualização fornecem uma ferramenta focada no tempo, atividades e nos aspectos de severidade/prioridade dos defeitos. A *System Radiography* tem como objetivo compreender a base de informações dos sistemas de *bug tracking* de uma forma geral. Esta visualização é útil para compreender como os defeitos são distribuídos nos softwares e seus componentes ao longo do tempo.

Knab e outros [Knab et al. 2009] propuseram uma abordagem de visualização para triagem de defeitos que fornece diversas visões significativas para explorar a qualidade da estimativa e o ciclo de vida dos defeitos relatados. A abordagem utiliza uma combinação de visualizações gráficas para investigar detalhes de defeitos individuais enquanto mantém o contexto dos dados fornecidos.

Hora e outros [Hora et al. 2012] apresentaram uma ferramenta para análise visual de repositórios de sistemas de *bug tracking* chamada *BugMaps*, que mapeia os *bugs* para defeitos nas classes em sistemas orientado a objetos. Esta ferramenta provê também diversas visualizações interativas para a tomada de decisão de gestores e *stakeholders* no desenvolvimento de software.

Uma técnica diferente de visualização de repositórios de sistemas de *bug tracking* foi proposta por Dal Sasso e Lanza [Dal Sasso and Lanza 2013, Dal Sasso and Lanza 2014] para representar uma visão refinada de um defeito. Para analisar os repositórios de sistemas de *Bug Tracking*, eles também propõem uma plataforma analítica visual, chamada *In*Bug*.

Yeasmin e outros [Yeasmin et al. 2014] propõem um protótipo que auxilia os desenvolvedores na revisão de defeitos dos projetos de software. Através de uma forma interativa, permite visualizar informações detalhadas referentes aos defeitos relatados usando a técnica de análise de assunto. Além disso, a fim de reduzir o tempo e os esforços dos desenvolvedores quando se estuda um defeito, o protótipo proposto também fornece uma visualização resumida de cada relatório de defeito.

Conforme os trabalhos de ferramentas de análise visual de sistemas de *bug tracking* discutidos anteriormente, em apenas um destes trabalhos é abordado a possibilidade de estender a ferramenta proposta para outros repositórios de sistemas de *bug tracking*.

A API desenvolvida neste trabalho se difere dos demais, pois proporciona uma arquitetura de software que possibilite o uso de futuras implementações para suportar a coleta de informações de repositórios de diferentes sistemas de *bug tracking*. Neste primeiro momento, a API proposta suporta o repositório do sistema de *bug tracking Redmine* [Lesyuk 2013].

5. Apresentando o BugInf

Nesta seção, é apresentado o *BugInf*, API livre que faz a coleta e integração com outras ferramentas de sofware livre para prover mecanismos de análise visual de sistemas de

bug tracking. É fornecida uma arquitetura de software que possibilite estender a análise visual para diferentes repositórios de sistemas de *bug tracking*. O objetivo é integrar diversas ferramentas livres para fornecer visões personalizadas através de um *dashboard* com a finalidade de apoiar o acompanhamento da evolução da qualidade do software e auxiliar na tomada de decisões dos gestores e times de desenvolvimento.

A figura 2 apresenta a arquitetura da API e seus respectivos módulos, em termos de seus componentes internos e sua relação com os sistemas de *bug tracking*, bem como suas integrações com as ferramentas livres *ElasticSearch* e *Kibana*.

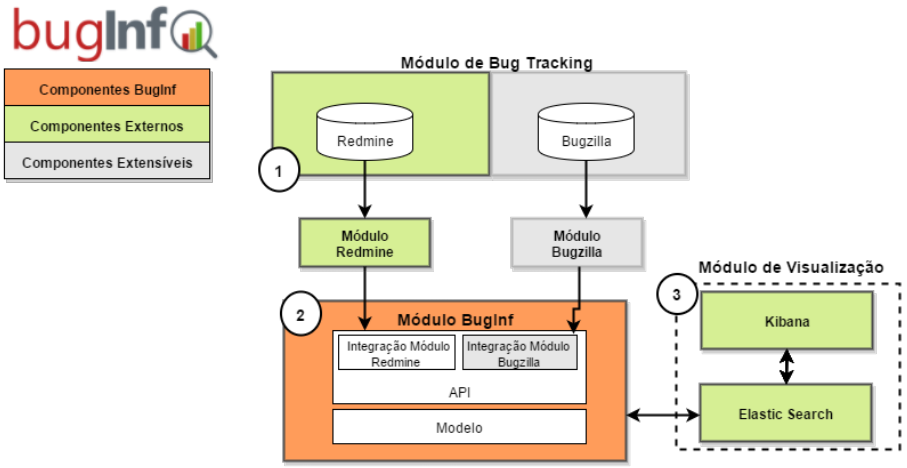

Figura 2. Arquitetura do BugInf

5.1. Arquitetura e Funcionamento

O *BugInf* foi estruturado em módulos. Para usar a API proposta, foi implementado, inicialmente, uma aplicação *standalone*, desenvolvida na linguagem de programação Java SE 7[2], que representa o módulo *Bug Tracking*. Este módulo tem como principal função recuperar e processar dados de repositórios de sistemas de *bug tracking*. Inicialmente, foi escolhido o *Redmine* [Lesyuk 2013] como *bug tracking* para implementação. A API usa como entrada a URL do repositório do sistema de *bug tracking* que será analisado. O módulo do *Redmine* implementado recupera e analisa todos os relatórios de defeitos em formato *JSON* [Crockford 2008] a partir do repositório de dados informado. Em seguida, ele preenche a parte correspondente do modelo de defeito definido no módulo *BugInf*.

O módulo *BugInf* é o principal, que disponibiliza a API desenvolvida. Ele é responsável por acionar o processo de coleta de dados dos repositórios do sistemas de *bug tracking* (implementado no módulo *Bug Tracking*). A coleta destas informações ocorre por meio de uma API de terceiros chamada *taskadapter* [3]. Ela encapsula o uso da API *RESTful* [Richardson and Ruby 2008] do *Redmine*, tratando o retorno das informações do formato *JSON* [Crockford 2008] para objetos da aplicação. Após este processo de coleta, é efetuado o preenchimento do modelo de defeitos com estas informações.

[2]http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase7-521261.html#jdk-7u76-oth-JPR

[3]https://github.com/taskadapter/redmine-java-api

Como cada ferramenta de *bug tracking* tem uma estrutura diferente, foi criado um modelo único que pudesse integrar estas informações. O modelo de defeitos criado é semelhante aos gravados em sistemas de *bug tracking*, como *Redmine*. Este modelo é composto pela descrição do problema, a sua prioridade (para corrigir o defeito), o membro da equipe que está resolvendo o defeito, *status* entre outros campos importantes para o entendimento do defeito. Na tabela 2, é descrito com maiores detalhes os atributos do modelo de defeito criado para prover as análises dos dados coletados e o mapeamento dos campos do *Redmine* com o modelo proposto.

Para o melhor entendimento de como foi projetada o *BugInf*, na figura 3 é apresentado os principais componentes da API. Os pacotes *facade* e *facade.impl* são responsáveis por definir as classes e interfaces de acesso à API. Já nos pacotes *service* e *service.impl* estão as classes que implementam a lógica de serviços e chamadas para as ferramentas. As classes que representam o modelo de defeitos e estrutura de projeto estão no pacote *model*. Os pacotes *type* e *constants* contém as definições de tipos enumerados e constantes que ajudam na estruturação do modelo proposto e do mapeamento com *ElasticSearch* e *Redmine*, descritos na tabela 2. As conversões para *JSON* e tratamento de datas são realizadas pelas classes do pacote *util*. As transformações do modelo do *bug tracking* para o modelo da API fica sob responsabilidade das classes do pacote *mapper*. Para execução do *BugInf*, foi desenvolvido uma aplicação console que dispara as chamadas da API (pacote *client*).

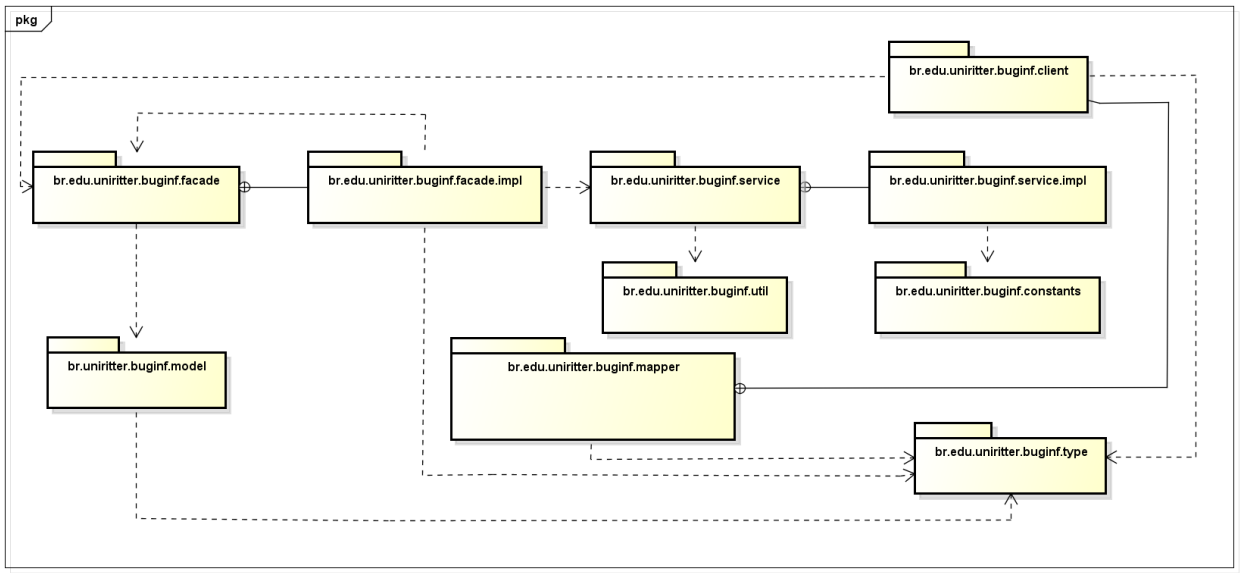

Figura 3. Componentes da API *BugInf*

Foram utilizados diversos padrões de programação orientada a objetos [Wu 2008], para facilitar uma possível expansão da integração com outros sistemas de *bug tracking*. Neste sentido, utilizou-se alguns padrões de projeto [Gamma et al. 1994] tais como, *Facade* para delegação das lógicas utilizadas pela API e *Factory* para criação de objetos conforme o tipo de *bug tracking*.

A partir deste momento, entra em ação o módulo de *Visualização*. Este módulo

Campo	Descrição	Redmine	BugInf
id	Identificador do defeito	-	-
descricao	Descrição resumida do defeito	-	-
status	Status que o defeito se encontra	*New, Needs Feedback, Confirmed, Resolved, Closed, Reopened*	Novo, Pendente de Feedback, Em Andamento, Reaberto, Resolvido, Não Definido
versao	Versão do sistema que foi encontrado o defeito	-	-
situacao	Situação que o defeito se encontra	*Open, Closed*	Aberto, Fechado
categoria	Identificador da categoria do sistema que foi afetada pelo defeito	*Accounts, UI, Database, Email Notifications, Issues, LDAP, SCM, etc.*	Interface, Plugins, Issues, Emails, Wiki, Controle de Versão, API, Texto, Autenticação, Banco de Dados, Performance, Componentes de Terceiros, Não Definida
atribuidoPara	Membro da equipe que está corrigindo o defeito	-	-
Prioridade	Representa a prioridade para o defeito ser corrigido	*Low, Normal, High, Urgent*	Baixa, Normal, Alta, Urgente, Imediata, Não Definida
dataCriacao	Data que o defeito foi criado	-	-
dataEncerramento	Data que o defeito resolvido	-	-
projeto/id	Identificador do projeto que foi afetado pelo defeito	-	-
projeto/nome	Nome do projeto que foi afetado pelo defeito	-	-
projeto/bugtracking	Identificador do bug tracking que o projeto está relacionado	-	-

Tabela 2. Descrição do modelo de defeitos definido para armazenar as informações usadas pela API

é composto pelas ferramentas *ElasticSearch* [4] e *Kibana* [5], esta última customizada para definição do *dashboard* (figura 4).

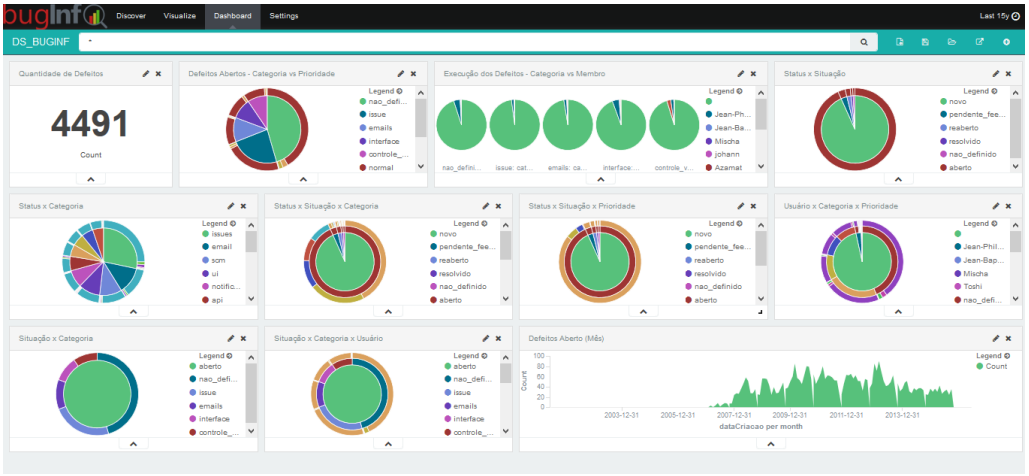

Figura 4. *Dashboard* criado no Kibana, representando o módulo de visualização da API BugInf

O *ElasticSearch* é um servidor de pesquisa baseado no *Apache Lucene*, uma *engine* de pesquisa *full-text* [Gormley and Tong 2014]. Ele armazena os dados em forma de documentos e disponibiliza-os no formato *JSON*. Ele pode ser usado para realizar pesquisas quase em tempo real dos documentos, para descobrir e analisar dados. Para interagir com *ElasticSearch*, tem-se o *Kibana*. Basicamente, o *Kibana* é uma interface web para analisar dados mantidos pelo *ElasticSearch*. Com isso, o *Kibana* ajuda os usuários a compreender os dados com ferramentas criadas para busca e criação de visualizações, permitindo a construção de *dashboards*. Estas ferramentas foram escolhidas por possuir o suporte nativo entre elas e a vantagem de suportar grande massa de dados. Com o suporte à escalabilidade do *ElasticSearch*, é possível suportar grande quantidade de informações dos repositórios de sistemas de *bug tracking* [Gormley and Tong 2014].

Quando o módulo *BugInf* realiza chamadas para o *ElasticSearch* [Gormley and Tong 2014], através de API*RESTFul* [Richardson and Ruby 2008], o *Kibana* usa estas informações para criar visualizações interativas customizadas pelo desenvolvedor para respectivas análises. Para a customização do *dashboard* no *Kibana*, foram definidas algumas métricas de acompanhamento, tais como quantidade de defeitos abertos, por prioridade, por membro e categoria, por período, entre outros.

5.2. Experimento e Resultados Preliminares

Para fins de validação, foi realizado uma implementação usando o projeto livre disponível do *Redmine*[6]. Este projeto possui mais de 8 anos, com aproximadamente 15 mil defeitos registrados, sendo mais de 4400 em aberto. Atualmente, 17 pessoas vem contribuindo para o projeto. Foi criado um *dashboard* (figura 4), composto de *portlets*, representando diferentes visualizações e agrupamento de dados. A figura 5 apresenta um histograma do número de defeitos ao longo do tempo.

[4]https://www.elastic.co/products/elasticsearch
[5]https://www.elastic.co/products/kibana
[6]https://www.redmine.org/projects/redmine

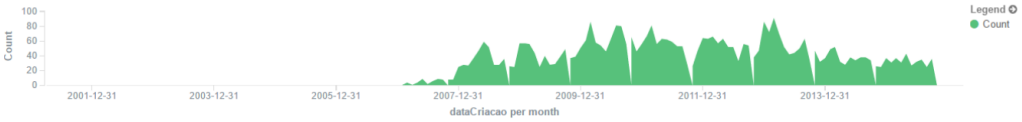

Figura 5. Histograma que apresenta o número de defeitos em uma linha do tempo

Com o mapeamento dos dados passados pelo *BugInf* para o *ElasticSearch/Kibana*, torna-se de fácil configuração e ajuste para cruzamento de dados, consultas diversas cruzando *status* de defeitos, com membros e categorias. Um exemplo é a quantidade de defeitos em relação a categoria e o membro do time de desenvolvimento que está executando sua correção é apresentado na figura 6. Este gráfico permite aos envolvidos do projeto visualizar de uma maneira geral de como as tarefas estão sendo executadas e, a partir disso, tomar decisões para, por exemplo, realocar a equipe para resolver as categorias de defeitos que são mais prioritárias para o projeto.

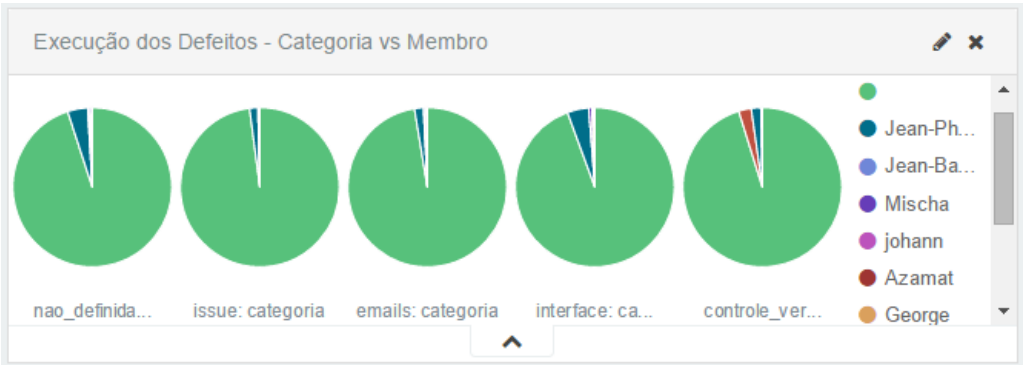

Figura 6. Gráfico que representa a quantidade de defeitos em execução em relação a categoria e o membro da equipe

Em um outro cenário de uso (figura 7), tem-se um gráfico que apresenta a quantidade de defeitos, filtrados por categorias do produto, com a devida prioridade por usuários (informações sobre categoria e prioridade estão definidas na tabela 2). Assim, com base nestes números, pode-se gerar algumas ações para corrigir estes defeitos, levando em consideração a sua prioridade, ajudando os usuários na sua organização. Também, é possível visualizar quais usuários estão mais sobrecarregados e, com isso, dividir as tarefas para execução.

A API proposta não se limita somente à integração de defeitos do *Redmine*, podendo ser estendido para outros sistemas de *Bug Tracking*. No *GitHub*[7] do *BugInf*, está disponível um tutorial de como estender a API para outros sistemas de *bug trackings*. Embora promissora, ainda necessita-se realizar mais experimentos com a API, seja com outros *bug trackings* como com projetos reais.

6. Considerações Finais

Neste trabalho, foi apresentado a API Livre denominada *BugInf*. Esta API provê mecanismos para automatizar a coleta de informações de sistemas de *bug tracking* e a realização

[7]https://github.com/diegojoseoschoski/buginf

Figura 7. Gráfico que apresenta a relação de usuários, categorias e prioridade a quantidade de defeitos em execução em relação a categoria e o membro da equipe

de análises visuais dos dados armazenados nos repositórios de sistemas de *bug tracking*. Isso, graças a integração com duas outras ferramentas de sofware livre, *ElasticSearch* e *Kibana*.

Estas visualizações personalizadas permitem apoiar o acompanhamento da evolução qualidade do software e auxiliar na tomada de decisão dos gestores e da equipe de garantia da qualidade. Identificou-se que, aos poucos, está crescendo o desenvolvimento de ferramentas para análise visual de repositórios de sistemas de *bug tracking*. Foi realizado um experimento inicial de uso da API com o repositório do *Redmine*, sendo possível analisar estas informações de forma visual e consolidada, apresentando um grande potencial para exploração do tema.

Para trabalhos futuros, pretende-se intensificar os experimentos, sobremaneira, com times de desenvolvimento de software. Também pretende-se definir novas métricas de defeitos e o desenvolvimento de uma nova integração da ferramenta para suportar outros sistemas de *bug tracking*, bem como, a criação de novas visualizações para complementar a API.

Referências

Bertram, D. (2009). *The social nature of issue tracking in software engineering*. PhD thesis, University of Calgary.

BT, M. (2016). Mantis. `https://www.mantisbt.org/`.

Bugzilla (2016). Bugzilla about. `https://www.bugzilla.org/about/`.

Cemin, C. (2001). *Visualização de informações aplicada à gerência de software*. PhD thesis, UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL.

Crockford, D. (2008). *JavaScript: The Good Parts: The Good Parts*. "O'Reilly Media, Inc.".

Dal Sasso, T. and Lanza, M. (2013). A closer look at bugs. In *Software Visualization (VISSOFT), 2013 First IEEE Working Conference on*, pages 1–4. IEEE.

Dal Sasso, T. and Lanza, M. (2014). In bug: Visual analytics of bug repositories. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pages 415–419. IEEE.

D'Ambros, M. and Lanza, M. (2006). Software bugs and evolution: A visual approach to uncover their relationship. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, pages 10–pp. IEEE.

D'Ambros, M., Lanza, M., and Pinzger, M. (2007). "a bug's life"visualizing a bug database. In *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*, pages 113–120. IEEE.

D'Ambros, M., Lanza, M., and Robbes, R. (2012). Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, 17(4-5):531–577.

Do Nascimento, H. A. and Ferreira, C. B. (2005). Visualização de informações–uma abordagem prática. In *XXV Congresso da Sociedade Brasileira de Computação, XXIV JAI. UNISINOS, S. Leopoldo–RS*.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software*. Pearson Education.

Gershon, N. and Eick, S. (1997). Information visualization. *Computer Graphics and Applications, IEEE*, 17(4):29–31.

Ghahrai, A. (2015). Top 8 open source bug tracking tools. `http://www.testingexcellence.com/top-8-open-source-bug-tracking-tools/`.

Gormley, C. and Tong, Z. (2014). *Elasticsearch: The Definitive Guide*. O'Reilly & Associates.

Hora, A., Anquetil, N., Ducasse, S., Bhatti, M., Couto, C., Valente, M. T., and Martins, J. (2012). Bug maps: A tool for the visual exploration and analysis of bugs. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pages 523–526. IEEE.

Janák, J. (2009). Issue tracking systems. *Brno, spring*.

Jean-Daniel Fekete, Jarke Van Wijk, J. S. C. N. (2008). The value of information visualization. *Lecture Notes in Computer Science*.

Knab, P., Fluri, B., Gall, H. C., and Pinzger, M. (2009). Interactive views for analyzing problem reports. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 527–530. IEEE.

Kumar, R., Gattaiah, D. Y., Shahi, S., and Nagendra, T. A. (2013). Improving software quality assurance using bug tracking system. *International Journal of Computer Science and Information Technologies*, 4.

Lata, K. and Sharma, S. (2015). Survey and study of various bug tracking and logging toolkits. *International Journal of Computer Applications*, 116(12).

Lesyuk, A. (2013). *Mastering Redmine*. Packt Publishing Ltd.

Mockus, A., Fielding, R. T., and Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346.

Redmine (2016). Redmine. `http://www.redmine.org/projects/redmine/wiki`.

Richardson, L. and Ruby, S. (2008). *RESTful web services*. "O'Reilly Media, Inc.".

Trac (2016). Trac wiki. `http://trac.edgewall.org/`.

Wang, X., Zhang, L., Xie, T., Anvik, J., and Sun, J. (2008). An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering*, pages 461–470. ACM.

Wu, C. T. (2008). *A Comprehensive Introduction to Object-Oriented Programming with Java*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

Yeasmin, S., Roy, C. K., and Schneider, K. A. (2014). Interactive visualization of bug reports using topic evolution and extractive summaries. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 421–425. IEEE.

Zimmermann, T., Premraj, R., Sillito, J., and Breu, S. (2009). Improving bug tracking systems. In *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 247–250. IEEE.