Kalibro Metrics: a multi-repository and multi-language source code analysis tool

Carlos Morais[1], Paulo Meirelles[1], Vinícius Daros[1], Fabio Kon[1]
[1]Department of Computer Science
Institute of Mathematics and Statistics
University of São Paulo, Brazil
{morais, paulormm, vkdaros, fabio.kon}@ime.usp.br

Carlos Santos Jr.[1,2]
[2]Horizon Institute
University of Nottingham, United Kingdom
carlos.denner@nottingham.ac.uk

***Resumo*—Source code metrics are not new, but they have not yet been fully explored in software development. For example, most metrics do not have known thresholds to guide their use by non-experts. Most metrics tools show isolated numeric values, which are not easy to understand because the interpretation of these values depends on the implementation context. Kalibro Metrics goal is to improve the readability of source code metrics. It allows a source code metric user to create a configuration of thresholds associated with qualitative evaluation, including comments and recommendations. Using these configurations, Kalibro shows metric results in a friendly way, helping software engineers to spot design flaws to refactor, project managers to control source code quality, and software adopters and researchers to compare specific source code characteristics across free software projects. These configurations are shared among its users via the Kalibro Web Service and a source code tracking network to enhance metric results interpretation.**

***Index Terms*—Software metrics, source code analysis, thresholds configuration, Web Service integration, source code tracking network, Free Software.**

I. Introduction

From a practical point of view, whatever the methodology, software development should be guided by two fundamental aspects to control the software quality in the long term: source code and test quality. Software testing does not mean to evaluate the source code quality. On the other hand, software source code metrics can help software engineers to observe the source code quality. Software Engineering requires the understanding of software, which is the result from the writing of source code. Also, software engineers and researchers need to analyze source codes to understand better software projects.

In the context of methodologies such as Agile Methods [1] and ones used by Free Software[1] [2] communities, source code is the main artefact of software development activities since features are constantly delivered to customers and users. Usually, software source code is written gradually and different developers make updates and improvements on an ongoing basis [3]. Thus, new features are inserted and bugs are resolved during software development and maintenance iterations.

An important observation on this process is that there is a significant gap between the numbers of lines of code which a software engineer reads and writes. Since developers read hundreds of lines of code (including their own past codes) to understand an implementation to change it or write more code, source code should be written to be read by other people [3].

In addition, software engineers need to make decisions about their codes when they are programming at the method and class level. The sum of these decisions influences the source code quality of their software [4]. To make the best decisions, they should monitor attributes from their source codes. For example, in a collaborative software development such as Free Software, source code organization and complexity influence the number of downloads and members from a Free Software project [5].

In this scenario, source code metrics can support the development of clean code, i.e., clear, flexible, and simple [3]. From an automated collection of metrics and an objective way to interpret their values, software engineers can monitor specific characteristics of their code, as well as detect troublesome scenarios to make decisions about the clarity, flexibility and simplicity of their codes at the method and class level.

However, even with the fact that source code metrics have been proposed since the 1970s [6], there is not a set of standard measures established. Moreover, little data are available or known to compare their effects, as well as there is not a systematic approach to use, interpret, and understand software metrics. Thus, software measurement, specially the use of source code metrics, are not fully explored in the software development [7]. Metric specialists are capable of using and understanding source code metrics, but all software engineers should know how to use these metrics to monitor and improve their source codes.

To make that easier, it is necessary for a tool to systematize the source code evaluation. We have identified, in particular, two limitations from current available tools, i.e., lack of the following features: (i) to collect automatically source code metrics values independent of the programming language; (ii) to interpret measurement results, associating them with source code quality. The first is important because software engineers and developers should be able to use the same tool for different languages, which would provide a similar set of metrics. The second takes a new approach how to analyze source code metrics values since an interpretation of them out

[1]In our work, we consider the terms "free software" and "open source software" equivalent.

of the software implementation context can not indicate and evaluate interesting source code attributes [6].

Since source code assessment tools frequently show their results as isolated numeric values for each metric, they are limited from the perspective of software engineers who are not metric specialists. Therefore, in this paper, we present Kalibro Metrics, a Free Software tool designed to incorporate any source code metric tool, extending it to provide easy to understand evaluation of the analyzed software quality. Kalibro Metrics allows a metric specialist user to specify a set of acceptance ranges to each metric provided by its integrated source code metric collector tools, as well as enabling the definition of customized metrics. With a set of thresholds and their respective interpretations, any software engineer can explore and better understand the use of source code metrics.

The remainder of this paper is organized as follows: Section **??** describes related work. Section **??** describes Kalibro architecture. Section **??** presents Kalibro features. Section **??** presents Kalibro use cases. Finally, Section **??** concludes the paper and discusses future work.

II. Projeto

O Portal do Software Público Brasileiro, SPB, inaugurado em 2007, na prática, é um sistema web que se consolidou como um ambiente de compartilhamento de projetos de software. Oferece um espaço (comunidade) para cada software. A comunidade é composta por fórum, notícias, chat, armazenamento de ae rquivos e downloads, wiki, lista de prestadores de serviços, usuários, coordenadores, entre outros recursos. Teve um crescimento expressivo contando, hoje, com mais de 60 comunidades de desenvolvimento e mais de 200.000 usuários cadastrados. O SPB abrange também, o 4CMBr que é o grupo de interesse voltado para soluções de tecnologia para municípios, o 5CQualiBr que é um grupo que trabalha para evoluir a qualidade do Software Público Brasileiro, o 4CTecBr, um portal destinado a colaboração no desenvolvimento de Tecnologias Livres, o Mercado Público Virtual que é um grupo de empresas e pessoas que prestam serviço nos softwares ofertados no Portal e o Ava- liaSPB que avalia a entrada dos softwares candidatos a software público. O ambiente do SPB não proporciona a integração com ambientes colaborativos externos, especialmente com redes sociais. A plataforma escolhida na ocasião da criação foi o framework OpenACS, que continua sendo utilizada na atual versão. A evolução do SPB foi comprometida desde 2009, quando framework OpenACS foi descontinuado. Com isso, não tendo versões lançadas a partir daquele ano. Por isso, hoje, é necessária a evolução para novas tecnologias, que tenham maior suporte das comunidades de desenvolvimento, utilize linguagens de programação com maior velocidade de desenvolvimento e permita a integração com ambientes colaborativos externos, em especial, redes sociais. Além disso, é preciso realizar a manutenção evolutiva das funcionalidades existentes e também o desenvolvimento de novas funcionalidades para o Portal do SPB . Um dos passos para a concretização de uma nova geração do Portal SPB é a integração de novas tecnologias, desde uma plataforma colaborativa até sistemas de controle de versão e monitoramento da qualidade do código-fonte, gerenciadas e apresentadas em uma plataforma integrada no back-end e, em especial, no front-end para que os usuários e as comunidades dos projetos tenham um conjunto de recursos para encontrarem os projetos, bem como colaborarem em torno de um sofware público. Mesmo com as limitações citadas, o Portal do Software Público Brasileiro teve em 2013 mais de 600 mil visitantes únicos, com mais de 1 milhão de visitas/acessos, gerando mais de 16 milhões de visitadas nas páginas, com um total de mais de 49 milhões de hits no Portal SPB. Avaliando apenas as comunidades dos projetos I3Geo, CAU, CACIC e Geplanes, houve mais de 15 mil downloads e 4 mil mensagens nos fórum. Essa amostra estatística ilustra bem o potencial do Software Público Brasileiro, bem como as expectativas de seus usuários e colaboradores para a evolução do Portal e do modelo em si.

III. Arquitetura

O sistema que irá atender a evolução e reformulação do Portal do Software Público Brasileiro constitui-se de diversos módulos e que irão comunicar-se entre si de forma organizada e integrada para suprir as necessidades do projeto.

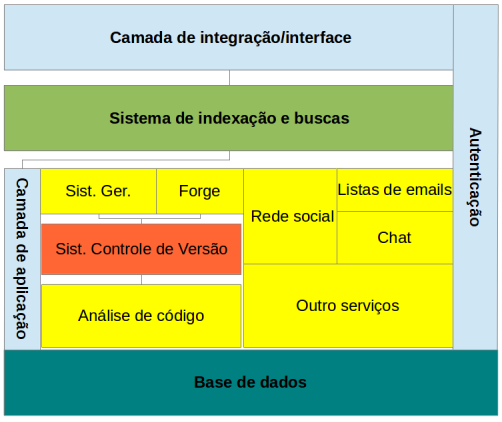

Figura 1. Proposta de arquitetura do Novo Portal do Software Público

As ferramentas que serão utilizadas para suprir essas necessidades serão:

- Para lista de e-mail estamos utilizando o Mailman na versão 2 que é um software gratuito para gerenciamento de discussão eletrônica de e-mail e listas e- newsletter.
- Para Chat estamos utilizando Punjab BOSH (XMPP) que é É uma interface HTTP cliente jabber. É um gerenciador de conexão BOSH que permite conexões de clientes persistentes para um servidor XMPP (protocolo de comunicação para mensagens orientadas a middleware baseado em XML)
- Para Plataforma de Buscas estamos utilizando Apache Solr que é uma plataforma de busca open source da Apache Lucene escrita em Java
- Para rede social estamos utilizando o Noosfero que é uma plataforma web livre para criação de redes sociais

com blog, e-Portifólios, CMS, RSS, discussão temática, agenda de eventos, galeria de imagens, chat, entre outros. Ele foi desenvolvido pela Cooperativa de Tecnologias Livres – Colivre 3 em 2007, sob a licença AGPL v.3, com a proposta de permitir ao usuário criar sua própria rede social personalizada, livre e autônoma.
- Para Forge para SVN estamos utilizando o Trac que é uma ferramenta open source para controle de mudança em projetos de desenvolvimento
- Para sistemas de controle de versão estamos utilizando SVN e Git que são ferramentas open-source para controle de mudança em projetos de desenvolvimento
- Para Forge para Git estamos utilizando o GitLab que é um software livre de colaboração de código online que utiliza a ferramenta de gerência de código-fonte Git
- Para sistema de gerenciamento estamos utilizando o Redmine que é uma aplicação web de gerenciamento de projetos que disponibiliza diversas ferramentas para auxiliar a gestão e manutenção de um projeto
- Para suporte a Single Sign On estamos utilizando o Mozilla Persona que foi desenvolvido pela Mozilla e permite o suporte a single sign on
- Para Sistema de Integração contínua estamos utilizando o Jenkins que é uma aplicação web de integração contínua de construção de projetos

Para integrar todas estas ferramentas estamos utilizando o Colab que nada mais é que uma plataforma de integração de ferramentas, o qual são integradas também as interfaces das ferramentas para que ao navegar o usuário tenha a sensação de estar navegando em uma única ferramenta.

Neste projeto notamos a necessidade de utilizar vários elementos que softwares livres já oferecem e não vimos sentido em refazer algo que já está pronto e disponível para ser utilizado. No início do projeto gastamos algum tempo estudando possíveis ferramentas que pudessem ser utilizadas para resolver os diferentes problemas e para resolver e chegamos a lista apresentada.

IV. Metodologia de Desenvolvimento

A Engenharia de Software tem evoluído suas práticas e metodologias em busca de padrões que regem o desenvolvimento de software de qualidade dentro dos escopos, custos e prazos desejados. O modelo dito tradicional tem como característica um conjunto grande e detalhado de documentação que deve, supostamente, ser utilizada ao longo de todo o ciclo de desenvolvimento. Entretanto, percebendo que os objetivos principais do desenvolvimento de software não estavam sendo alcançados, alguns líderes da indústria e academia começaram a adotar métodos mais simples de trabalho que apresentaram melhores resultados em projetos de software. Em 2001, líderes que estavam desenvolvendo projetos fora dos padrões industriais se reuniram para trocar experiências e trabalhos. Este grupo se tornou a Aliança de Desenvolvimento Ágil e escreveram o Manisfesto Ágil que apresenta os princípios e valores que o grupo considera ser determinante para o desenvolvimento de software. Neste sentido, os métodos de desenvolvimento ágeis de software são métodos que implementam os seguintes valores:

- Indivíduos e interações acima de processos e ferramentas;
- Software operante acima de documentações grandes e completas;
- Colaboração do cliente acima de negociações contratuais;
- Responder à mudanças acima de seguir a um planejamento.

Esse conjunto de valores não descartam a importância dos elementos citados à direita das sen- tenças, mas evidenciam que estes são menos importantes diante dos primeiros elementos citados. Em outras palavras, apesar da documentação ser importante, o foco principal deve estar na entrega de software de valor para o cliente e na interação e a consequente comunicação entre as pessoas. Além disso, os métodos ágeis exaltam a simplicidade, feedback contínuo e adaptação à mudanças que podem ser obitidos a partir de comunicação face à face, qualidade de código e entrega contínua de software.

Dada a oportunidade de adoção de métodos ágeis no desenvolvimento do presente trabalho, a metodologia utilizada é baseada em uma combinação das metodologias Scrum e Extreme Programming. Destacam que XP e Scrum complementam um ao outro bem, com o XP provendo suporte para aspectos mais técnicos enquanto o Scrum provê práticas e técnicas para gerenciamento, planejamento e acompanhamento. Assim,com base nas experiências destacadas em [Schwaber and Beedle, 2001] e 21 [Fitzgerald et al., 2006], na motivação de adoção de métodos ágeis no desenvolvimento de software moderno, serão apresentadas os métodos XP e Scrum, suas principais características e práticas que serão utilizadas no desenvolvimento do presente projeto.

V. Escolha da equipe

A equipe é formada majoritariamente por alunos de graduação do curso de Engenharia de Software da Universidade de Brasília, conta ainda com dois ex-alunos formados, alunos de mestrado que trabalham exclusivamente com o design além de professores orientadores.

Devido a esta formação a equipe não consegue estar sempre trabalhando junta pois cada um possui seu horário de aula e adéqua as horas de contribuição ao projeto a este horário de aulas, dessa forma não conseguimos utilizar os métodos ágeis na íntegra.

Esta equipe maior está dividida em duas equipes menores onde uma dessas equipes está trabalhando com o Nosfero e a outra equipe está trabalhando com Colab e as integrações das demais ferramentas. A equipe do Noosfero está desenvolvendo um plugin com novas funcionalidades para o Portal do Software Público. A equipe do Colab ainda está trabalhando com configurações das ferramentas pois está sendo integrado o Redmine e o Gitlab com o Colab e está exigindo maiores esforços de infra-estrutura.

Para manter o controle das atividades do projeto e evitar os ruídos de comunicação temos uma lista de e-mail com todos os integrantes do projeto, onde foi criado o hábito de enviarmos um e-mail ao final do dia com as atividades que

desenvolvemos. Temos também um dia da semana em que todos estão no laboratório onde fazemos um stand up para alinharmos a situação das equipes. Utilizamos ainda o Redmine para fazer o gerenciaento do projeto, onde são escritas histórias de usuários e histórias técnicas e suas respectivas tarefas. Estamos trabalhando em sprints/ciclos de duas semanas, em média, e releases de 4 meses, sendo que o projeto tem duração programada de 3 anos, sendo ao total 7 releases e 58 sprints.

VI. Relato de Experiência

Foi aplicado um questionário na equie de desenvolvimento do Novo Portal do Software Público onde os mesmos poderiam expressar o quanto estão aprendendo com o projeto e o quanto o projeto está lhe ajudando na sua formação.

O questioário foi respondido por 17 integrantes do projeto que têm entre 20 e 26 anos estando entre o 5º e o 10º semestre.

A primeira questão pretendia averiguar o nível de conhecimento dos entrevistados, em uma escala de 1 a 5, em relação às ferramentas que estão sendo utilizadas no projeto. Como respostas obtivemos que 63% tem nível 3, 19% nível 2 e 19% nível 4.

A segunda questão pretendia averiguar o quanto o projeto está contribuindo para a formação do entrevistado. Em uma escala de 0 a 5, 69% dos entrevistados responderam "5 - Contribui muito, mais que determinadas disciplinas", 19% dos entrevistados responderam "4 - Contribui, no mesmo nível de muitas disciplinas"e 13% dos entrevistados responderam "3 - Contribui, da mesma forma que um estágio convencional"

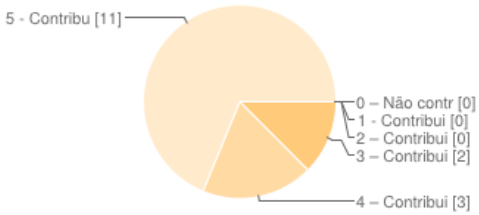

Figura 2. Respostas do nível de contribuição do projeto na sua formação

A terceira questão trata, também em uma escala de 0 a 5, do quanto o projeto atrapalha de sua performance nas disciplinas da graduação. 19% dos entrevistados responderam "3 - Atrapalha minha graduação da mesma forma que um estágio", 50% dos entrevistados responderam "2 - Atrapalha moderadamente", 19% dos entrevistados responderam "1 - Atrapalha pouco, menos que um estágio"e 13% dos entrevistados responderam "0 Não atrapalha em nada na minha graduação".

A quarta questão averigua o quanto os conhecimentos adiquiridos durante a execução do projeto ajuda os entrevistados nas disciplinas da graduação, em uma escala de 0 a 5. 7% dos entrevistados responderam "5 - Ajuda muito, sempre utilizo os conhecimentos adiquiridos nas disciplinas da graduação ", 47% respondeu "4 Ajuda muito, utilizo os conhecimentos com frequência nas disciplinas da graduação", 27% respondeu "3

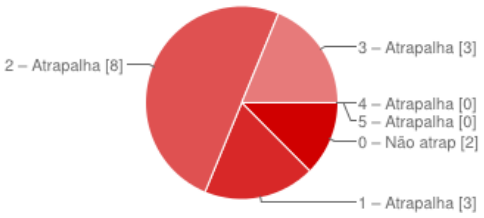

Figura 3. Respostas do nível de performance nas disciplinas de graduação

- Ajuda, utilizo os conhecimentos nas disciplinas da graduação"13% respondeu "2 - Ajuda moderadamente, as vezes utilizo os conhecimentos adquiridos no projeto"e 7% respondeu "1 – Ajuda um pouco, utilizo pouco os conhecimentos do projeto"

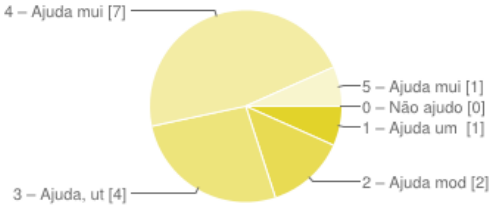

Figura 4. Respostas do nível de conhecimento adquirido

E a quinta e última questão perguntou o quanto o entrevistado acredita que o projeto NSPB vai lhe dar de experiência para o mercado de trabalho, ainda em uma escala de 0 a 5. 38% respondeu "5 O projeto me tornará experiente para o mercado de trabalho", 38% respondeu "4 - O projeto me dará muita experiência para o mercado de trabalho"e 25% respondeu "3 - O projeto me dará uma boa experiência para o mercado de trabalho".

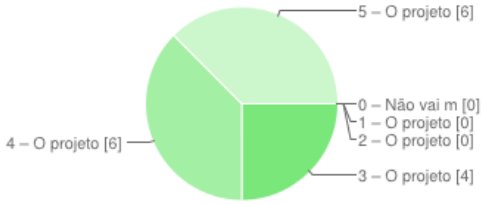

Figura 5. Respostas do nível de experiência para o mercado de trabalho

Referências

[1] K. Beck, *Extreme Programming Explained*. Addison Wesley, 1999.

[2] FSF, "The free software definition," Free Software Foundation Web Site, April 2011. [Online]. Available: http://www.gnu.org/philosophy/free-sw.html

[3] R. C. Martin, *Clean Code - A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

[4] K. Beck, *Implementation Pattens*. Addison Wesley, 2007.

[5] P. Meirelles, C. Santos Jr., J. Miranda, F. Kon, A. Terceiro, and C. Chavez, "A study of the relationships between source code metrics and attractiveness in free software projects," *Software Engineering, Brazilian Symposium on*, vol. 0, pp. 11–20, 2010.

[6] E. E. Mills, “Software Metrics,” Software Engineering Institute, SEI - Carnegie Mellon University, Tech. Rep., 1988.
[7] E. Tempero, “On Measuring Java Software,” in *ACSC ’08: Proceedings of the Thirty-First Australasian Conference on Computer Science*, vol. 74. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2008, pp. 7–7.