Kalibro Metrics: um serviço para monitoramento e interpretação de métricas de código-fonte

Carlos Morais, Paulo Meirelles, Eduardo Morais

[1]Centro de Competência em Software Livre
Instituto de Matemática e Estatística
Universidade de São Paulo

{morais,paulormm,emorais}@ime.usp.br

Resumo. *Métricas de código-fonte não são novidade, mas ainda não têm sido bem exploradas no desenvolvimento de software. A maioria das ferramentas de métricas mostra valores numéricos isolados, que não são fáceis de entender porque a interpretação desses valores pode depender do contexto de implementação.*
O objetivo do Kalibro Metrics é melhorar a legibilidade de métricas de código-fonte, difundindo seu uso. Ele permite ao usuário criar configurações de intervalos associados a avaliações qualitativas, incluindo comentários e recomendações. Usando essas configurações, Kalibro mostra resultados de métricas de modo amigável, ajudando: arquitetos de software a detectar falhas de projeto; gerentes de projetos a controlar qualidade de código-fonte; desenvolvedores de software e pesquisadores a comparar características específicas do código-fonte de vários projetos. Essas configurações podem ser compartilhadas e evoluir.

1. Introdução

Qualquer que seja a metodologia de desenvolvimento, monitorar a qualidade do software é fundamental. Dentre as inúmeras características que fazem um bom software, várias delas podem ser percebidas no código-fonte, e algumas são exclusivas dele. Por exemplo, código compilado pode ser analisado, mas características como organização e legibilidade são perdidas; mesmo uma bateria de testes com ótima cobertura só apresenta informação sobre o funcionamento atual, não refletindo manutenibilidade, modularidade, flexibilidade e simplicidade. Nesse contexto, as métricas de código-fonte completam as outras abordagens de monitoramento da qualidade do software.

Engenheiros de software e pesquisadores precisam analisar códigos-fonte para entender melhor projetos de software. Com o sucesso dos Métodos Ágeis [Beck 1999] e do Software Livre [FSF 2011], o código-fonte é destacado como tendo papel central nas atividades de desenvolvimento de software. As funcionalidades são entregues constantemente aos clientes e usuários, portanto o código-fonte é escrito gradualmente e diferentes desenvolvedores fazem atualizações e melhorias continuamente [Martin 2008]. Assim, novas funcionalidades são inseridas e erros corrigidos durante as iterações de desenvolvimento e manutenção.

Desenvolvedores tomam decisões sobre o código enquanto estão programando. Em conjunto, essas decisões influenciam na qualidade do código [Beck 2007]. Observar os atributos do código que está desenvolvendo pode auxiliar na tomada dessas decisões.

No contexto dos projetos de software livre, há estudos que mostram que a motivação para se envolver com um projeto de software livre é criticamente afetada pela modularidade do seu código [Baldwin and Clark 2006] e sua organização e complexidade influenciam o número de downloads e membros [Meirelles et al. 2010].

De modo geral, existe uma diferença significativa, tanto em tempo gasto como em quantidade de linhas que um desenvolvedor lê e escreve. Como é preciso entender uma implementação lendo código (inclusive escrito no passado pelo próprio leitor) para poder melhorará-la ou incrementá-la, código-fonte deve ser escrito para ser entendido por pessoas, mais do que pela máquina [Martin 2008]. Nesse cenário, métricas de código-fonte podem auxiliar no desenvolvimento de código claro, simples e flexível. A partir de uma coleção de métricas coletadas automaticamente e de uma forma objetiva de interpretar seus valores, engenheiros de software podem monitorar características específicas do seu código - assim como implementações problemáticas - para tomar melhores decisões ao (re)escrevê-las.

Métricas de código-fonte foram propostas desde quando os primeiros conceitos da engenharia de software surgiram. As métricas pioneiras foram rapidamente absorvidas pela indústria, que as usa massivamente. Métricas de complexidade e tamanho são as mais usuais, por exemplo LOC (linhas de código) e seus derivados (erros/LOC, LOC/tempo). Essas métricas são reconhecidamente limitadas (e até perigosas) quando usadas isoladamente. Mas mesmo com o avanço da pesquisa em métricas desde então, a indústria continua usando as mesmas métricas simplistas e de forma isolada [Fenton and Neil 1999].

Especialistas em métricas podem ser capazes de entender e usar métricas, mas todos desenvolvedores deveriam saber como usá-las para monitorar e melhorar seu código. Para facilitar isso, é necessária uma ferramenta que sistematize a avaliação do código-fonte. As ferramentas de coleta de métricas de código-fonte possuem, em geral, uma ou mais dessas carências:

- Associação entre resultados numéricos e forma de interpretá-los. Ferramentas de métricas frequentemente mostram seus resultados como valores numéricos isolados para cada métrica.
- Flexibilidade nessa interpretação. Algumas ferramentas mostram indicadores binários (bom ou ruim) para os valores das métricas, mas não há forma de configurar os valores de referências.
- Flexibilidade para funcionar com diferentes linguagens. As ferramentas que não pecam nos quesitos anteriores foram desenhadas especificamente para uma só linguagem de programação.
- Capacidade de estender seu uso com novas métricas, além das embutidas na ferramenta.

Este trabalho apresenta Kalibro Metrics, um software livre em forma de serviço Web, projetado para se conectar a qualquer ferramenta de coleta de métricas de código-fonte, a estendendo para fornecer avaliação de fácil entendimento para a qualidade do software analisado.

Kalibro Metrics permite que um especialista em métricas (ou qualquer usuário) especifique conjuntos de intervalos de aceitação para cada métrica fornecida pela ferramenta base, além de possibilitar a criação de novas métricas, combinando as básicas. Com isso pretende-se difundir o uso de métricas de código-fonte, pois com um conjunto

de intervalos e suas respectivas interpretações, qualquer desenvolvedor pode explorá-las e melhor entendê-las.

A Seção 2 descreve os requisitos que os autores acreditam ser essenciais em uma ferramenta que vise a difundir o uso de métricas de código-fonte. O não-preenchimento desses requisitos nas ferramentas estudadas irá justificar a escolha de criar o Kalibro Metrics. A Seção 3 descreve o Kalibro Metrics, como sua arquitetura foi desenhada para facilitar o preenchimento desses requisitos e apresenta um caso de uso do Kalibro.

2. Requisitos

A indústria de desenvolvimento de software adotou métricas simples - como linhas de código e contagem de defeitos - porque elas são fáceis de entender e de coletar [Fenton and Neil 1999]. Deseja-se um software que colete métricas e associe seus resultados a alguma interpretação, tornando-as mais fáceis de entender. Ao constatar dificuldades nas ferramentas de métricas atuais, foram estabelecidos os seguintes requisitos para um software que vise a difundir o uso de métricas de código-fonte:

1. **Intervalos de aceitação:** Deve suportar múltiplos intervalos para fornecer diferentes interpretações sobre os valores das métricas. Por exemplo, usando o Eclipse Metrics [Walton 2011] é possível configurar valores mínimo e máximo (um intervalo de aceitação) para cada métrica que ele fornece, e associar uma dica de correção quando o resultado para um método/classe/pacote cai fora desse intervalo. Os intervalos devem ser configuráveis, pois as métricas em geral não possuem valores de referência absolutos ou universalmente aceitos. Os valores ideais podem variar de acordo com vários fatores, como a linguagem e o domínio de aplicação.
2. **Extensível:** Não deve se limitar a apenas uma linguagem de programação, nem a um conjunto pré-definido de métricas. Deve fornecer interface clara para adicionar suporte a novas linguagens de programação, pois isso pode atrair uma gama maior de usuários em potencial para sugerir intervalos de acordo com suas experiências em linguagens específicas. Além disso, o usuário deve ser capaz de configurar o conjunto de métricas que deseja analisar. Muitos coletores de métricas calculam e exibem o resultado de todas as métricas de seu repertório, o que consome mais recursos que o necessário (i.e. tempo e memória) e polui a exibição. Deve haver suporte para inclusão de novas métricas, a partir de diferentes coletores de métricas ou a partir das métricas existentes. O índice de manutenibilidade [Vandoren 1997] é um exemplo de métrica criada como combinação de métricas básicas. Outro exemplo é a complexidade estrutural, combinação de métricas de acoplamento e coesão, que teve sua influência sobre atratividade de projetos de software livre evidenciada [Meirelles et al. 2010].
3. **Comparação entre projetos:** A ferramenta deve dar suporte a comparação de características específicas do código-fonte de vários projetos através de métricas, dando suporte a pesquisadores e ajudando nas decisões de adotantes de software.
4. **Software Livre mantido ativamente:** A ferramenta deve ser software livre, disponível sem restrições e mantida ativamente. Assim, qualquer um pode contribuir com seu desenvolvimento e modificá-la de acordo com suas necessidades. Isso também permite que pesquisadores repliquem completamente estudos e resultados.

Ferramenta	Linguagens	Req. 1	Req. 2	Req. 3	Req. 4
Analizo	C, C++, Java	Não	Sim	Não	Sim
CCCC	C++, Java	Não	Não	Não	Sim
Checkstyle	Java	Sim	Não	Não	Sim
CMetrics	C	Não	Sim	Não	Sim
Cscope	C	Não	Não	Não	Sim
JaBUTi	Java	Não	Não	Não	Sim
MacXim	Java	Não	Não	Não	Sim
Metrics	Java	Sim	Não	Não	Sim

Tabela 1. Ferramentas existentes versus requisitos definidos

Durante a pesquisa e desenvolvimento, os autores estudaram ou utilizaram 8 ferramentas de software livre [1] para análise de código-fonte. Na Tabela 1, as ferramentas estudadas são comparadas com os requisitos. Percebe-se que nenhuma delas preenche todos.

Dessa carência vem a decisão de criar o Kalibro Metrics, um serviço responsável pelo suporte à interpretação das métricas. Foi definida uma interface de conexão simples para delegar a coleta de valores para ferramentas de análise de código-fonte existentes. Dessa forma, Kalibro pode se conectar com as ferramentas mais interessantes para o contexto de cada software analisado.

Como primeira ferramenta base, a Analizo [Terceiro et al. 2010] foi escolhida pelos seguintes motivos: capacidade de extensão para suporte a outras métricas/linguagens, pois sua arquitetura é baseada em extratores especializados; grande eficiência da análise de código C, C++ e Java, linguagens mais populares em projetos de software livre; bom número e variedade de métricas embutidas; saída fácil de ser analisada automaticamente (formato YAML).

3. Kalibro Metrics

Kalibro foi desenvolvido na linguagem Java, de forma modular e desacoplada, para facilitar modificações, incrementos e incorporações de outras ferramentas e diferentes interfaces de usuário. Toda a lógica, processamento e persistência foram isolados em um núcleo independente de interface com o usuário que chamamos de KalibroCore. Ele inclui uma fachada [Gamma et al. 1994] para acessar toda sua funcionalidade, a classe `Kalibro`. Assim, as únicas classes que são usadas fora do KalibroCore são `Kalibro` e suas dependências de interface. Foi desenvolvida uma interface Web Service (Kalibro Service) para acessar as funcionalidades do Kalibro remotamente. A Figura 1 mostra as duas implementações da fachada e como elas fornecem as funcionalidades do Kalibro.

Para interagir inicialmente com o Kalibro, foi desenvolvida uma interface Swing, o Kalibro Desktop. O Kalibro Desktop se comunica com a fachada do KalibroCore, assim o usuário pode configurá-la como aplicativo independente ou como cliente de um

[1] `analizo.org` `cccc.sourceforge.net` `checkstyle.sourceforge.net` `tools.libresoft.es/cmetrics` `cscope.sourceforge.net` `ccsl.icmc.usp.br/pt-br/projects/jabuti` `qualipso.dscpi.uninsubria.it/macxim` `metrics.sourceforge.net`

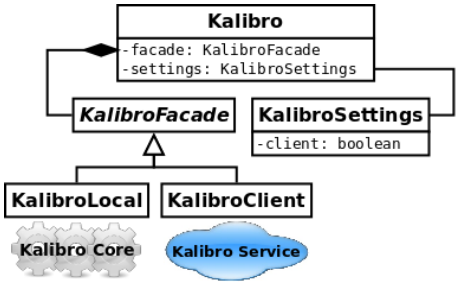

Figura 1. Fachada pode ser local ou remota

Kalibro Service remoto, variando apenas a implementação da fachada. Quando usado como cliente, a instalação completa (banco de dados, coletor, etc) só precisa ser feita no servidor e vários terminais podem utilizá-la com pouca configuração.

Posteriormente, o desenvolvimento do Kalibro Metrics foi associado ao desenvolvimento da plataforma de monitoramento e visualização de métricas Mezuro[2]. O Mezuro é uma rede sócio-técnica que tem o objetivo de ser um ambiente aberto e colaborativo de avaliações de código-fonte e aprendizado do "estado-da-prática" dos projetos de software livre. A rede Mezuro é construída sobre a plataforma de redes sociais Noosfero[3], através de um plugin que se comunica com o Kalibro Service.

Enfim, tanto o Kalibro Desktop como o Mezuro servem como camada de visualização dos resultados e funcionalidades do Kalibro. Essa arquitetura está representada na Figura 2.

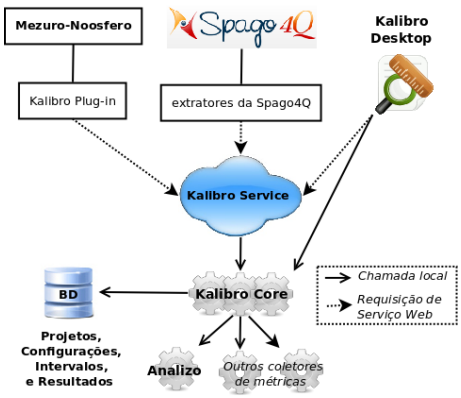

Figura 2. Diagrama de interações

As subsessões seguintes correspondem aos requisitos estabelecidos e como o Kalibro foi desenhada para atendê-los. Quando for preciso ilustrar os conceitos da arquitetura, mostraremos capturas de tela, ora do Kalibro Desktop, ora do Mezuro. Entretanto, não se pretende defender a forma como as métricas são visualizadas nesses aplicativos. Visualização de métricas é em si um campo de pesquisa. Os esforços no desenvolvimento do Kalibro concentram-se na avaliação qualitativa de métricas de código-fonte, não no *front-end*.

[2] `mezuro.org`
[3] `noosfero.org`

3.1. Configuração de intervalos

Adotamos o conceito de configuração de métricas. Uma configuração é um conjunto de métricas, cada métrica contendo um conjunto de intervalos. Essa configuração é criada e personalizada pelo usuário. Cada intervalo possui, além de início e fim, nota, cor para exibição e comentários pertinentes para quando o valor da métrica estiver dentro dele (veja a Figura 3). Dessa forma, é possível associar uma avaliação qualitativa ao valor obtido como resultado do cálculo da métrica.

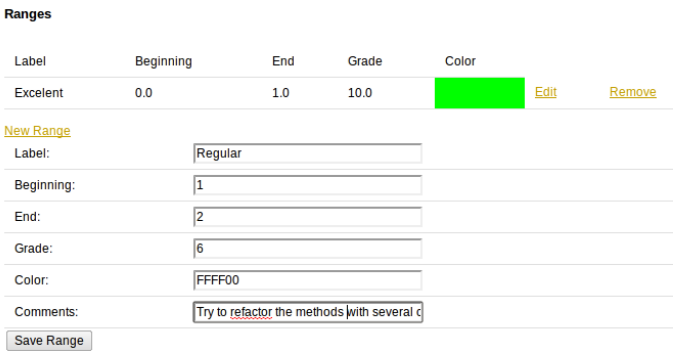

Figura 3. Editando um intervalo

Quando um projeto de software é analisado, apenas os resultados das métricas contidas na configuração associada são calculados, com seus valores associados aos intervalos correspondentes. A Figura 4 mostra resultados associados à avaliação qualitativa (configuração de intervalos). O usuário pode criar várias configurações e utilizá-las para atender demandas específicas de cada projeto.

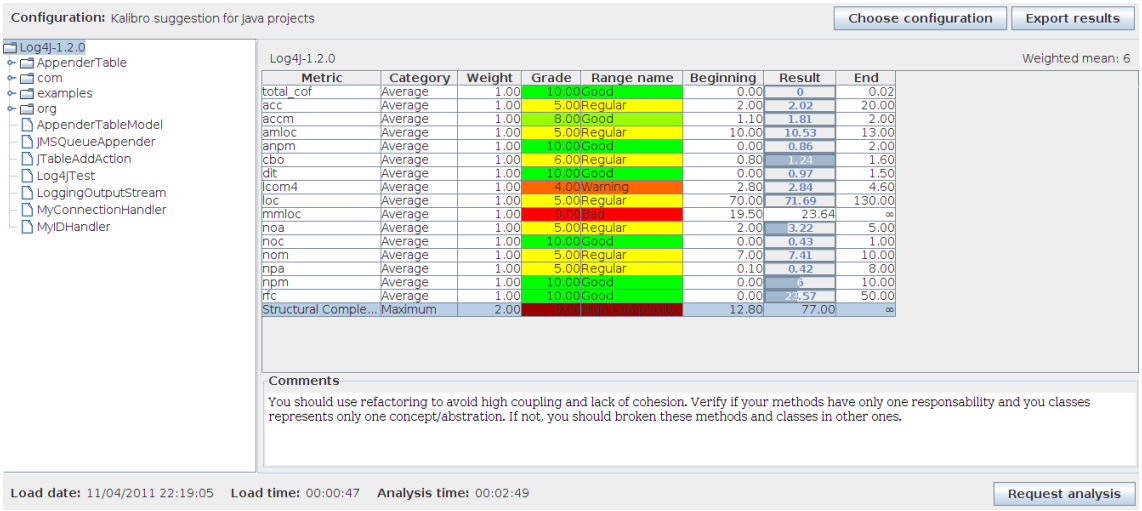

Figura 4. Resultados avaliados por uma configuração

3.2. Extensível

Kalibro Metrics foi projetado para ser facilmente conectado a diferentes ferramentas de coleta de métricas de código-fonte. A interface `MetricCollector` é como o Kalibro interage com as ferramentas coletoras de métricas (veja figura 5). O método

`getBaseTool` devolve uma descrição da ferramenta base, com as métricas suportadas, e o método `collectMetrics` coleta os resultados das métricas. Duas implementações dessa interface foram desenvolvidas, uma para o Analizo, e outra, num esforço mais recente, para o CheckStyle[4], que está em fase de testes. É essa interface que deve ser implementada para cada ferramenta com a qual o Kalibro se conecte. O usuário pode, desse modo, escolher métricas de coletores específicos de acordo com as características de seu projeto.

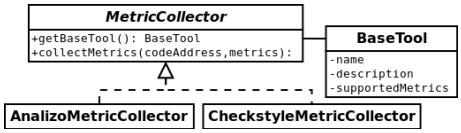

Figura 5. Interface para coleta de métricas

O outro quesito da extensibilidade é a capacidade de criar novas métricas a partir das existentes. Dividimos então as métricas utilizadas no Kalibro em métricas nativas e compostas. As métricas nativas são as fornecidas pelos coletores e as compostas são criadas a partir das nativas através de scripts. Após coletar os resultados das métricas nativas, Kalibro calcula estatísticas e valores das métricas compostas. A Figura 6 mostra a criação, pelo usuário, da métrica SC (complexidade estrutural) a partir das métricas LCOM4 (coesão) e CBO (acoplamento).

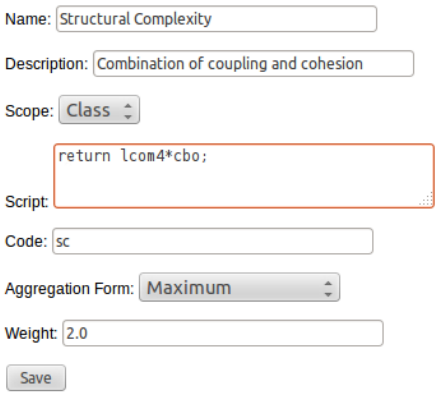

Figura 6. Criação de uma métrica composta

Resolvemos integrar ao Kalibro o download automático de código-fonte a partir de seu repositório por diversas razões. Por amigabilidade, pois o usuário precisa apenas indicar o endereço do código ao invés de baixá-lo e depois executar o Kalibro. Para dar suporte è utilização do Kalibro como serviço Web, pois enviar um endereço para o serviço remoto é muito mais conveniente que anexar o código-fonte. Para dar suporte a pesquisas que envolvem vários projetos, facilitando o download para compará-los. Finalmente, para facilitar o acompanhamento da evolução dos resultados das métricas de um projeto ao longo do tempo.

Kalibro carrega código-fonte de um repositório executando chamadas de sistema. Para adicionar um novo tipo de repositório basta estender a classe abstrata

[4]checkstyle.sourceforge.net

`ProjectLoader` e resgistrar o novo tipo de repositório (ver Figura 7). O principal método dessa classe é o `load`, que recebe um repositório e executa os comandos necessários para baixar ou atualizar o código. Esses comandos são advindos das subclasses que implementam o método abstrato `getLoadCommands`. Dessa forma, o Kalibro também é extensível para diversos tipos de repositório. Já implementamos o suporte a Bazaar, CVS, Git, Mercurial, Subversion, Tarball e Zip.

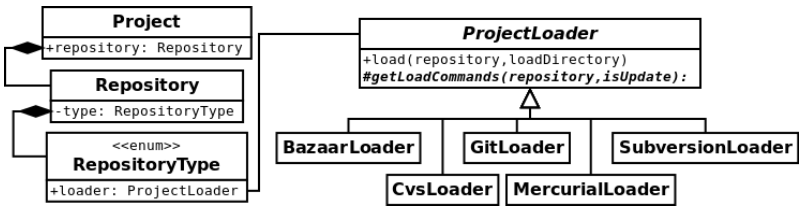

Figura 7. Arquitetura do carregadores de código

3.3. Comparação entre projetos

Além do conjunto de intervalos, cada métrica de uma configuração possui um peso. Com a nota dos intervalos e o peso de cada métrica, Kalibro calcula uma nota geral do projeto, pacote, classe ou método, com uma média ponderada. Desse modo, além de ser uma forma de interpretar resultados de métricas, as configurações também servem como critério específico de avaliação e comparação entre projetos de software. Na Figura 4 a nota do projeto aparece no canto superior direito da tela de resultados.

As configurações de métricas são a parte mais importante da flexibilidade do Kalibro. A configuração utilizada por um gerente de projeto para acompanhar a qualidade do código pode sofrer melhorias ao longo do tempo. O Mezuro servirá de plataforma para o compartilhamento dessas configurações, facilitando uma discussão em torno delas e sua subsequente evolução.

3.4. Software Livre

Kalibro Metrics é software livre, distribuído sob a licença GPL (GNU Lesser General Public License) versão 3. Seu código-fonte, assim como pacotes binários, manuais e tutoriais podem ser encontrados em `http://kalibro.org`. Sua arquitetura foi desenhada pensando em modularidade e flexibilidade, facilitando o recebimento de contribuições da comunidade de software livre. A seguir, outras características que torna seu código receptível a contribuições.

O download e a análise de projetos é executada de forma assíncrona. Ouvintes podem ser registrados para receber notificações quando o projeto passa por cada uma das fases de processamento (atualização do código, execução dos coletores e análise dos resultados). Ao usar o Kalibro como cliente do Kalibro Service, um adaptador especial faz *polling* do estado do(s) projeto(s) e notifica os ouvintes, encapsulando detalhes de conexão com o serviço.

Todo tratamento de threads é encapsulado em um só pacote. É possível criar tarefas e executá-las assincronamente, adiocionando um ouvinte à tarefa. O ouvinte recebe um relatório que informa se a tarefa completou normalmente, mostrando a exceção capturada se for o caso. As tarefas possuem métodos para execução de forma síncrona,

assíncrona, periódica e com *timeouts*. Também existe um tipo de tarefa específico para executar chamadas de sistema e salvar ou capturar sua saída.

Kalibro usa JPA - Java Persistence API [Biswas and Ort 2006] - para persistir suas entidades. Com pouca configuração, pode ser instalado para usar qualquer banco de dados com suporte a JPA.

4. Casos de usos

Kalibro Metrics teve sua primeira versão estável lançada em dezembro de 2010 e já tem sido usada por outros grupos de pesquisa. Kalibro Metrics foi integrado à QualiPSo Quality Platform através da Spago4Q[5]. O projeto QualiPSo começou em 2007, limitado a análise de projetos Java. Em 2010, para analisar C e C++, nós integramos o Kalibro Metrics via Kalibro Service dentro do projeto QualiPSo.

A Figura 8 apresenta como a Spago4Q se comunica com o Kalibro Service. A Spago4Q coleta resultados a partir de extratores. No caso do Kalibro Service, os extratores usam sua fachada, simplificando a interação. O extrator de upload lê requisição da fila da Spago4Q e envia requisições para o Kalibro processá-las. O extrator de análise consulta o serviço e fornece os valores de volta para a Spago4Q.

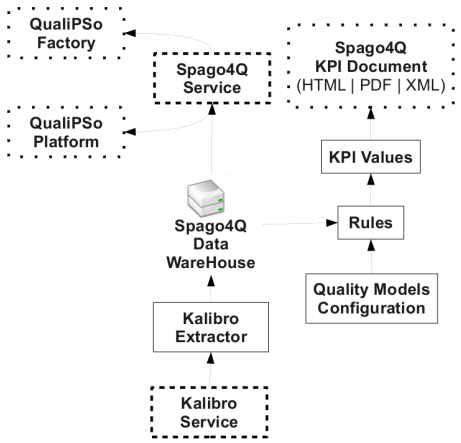

Figura 8. Integração com o Kalibro no QualiPSo.

Depois do projeto QualiPSo, o Kalibro Metrics foi associado ao desenvolvimento do Mezuro. Isso porque, da mesma forma que arquitetos de engenharia civil estudam durante sua formação as obras clássicas e reconhecidamente bem projetadas; e os escritores literários constroem seus conhecimentos e definem estilos aos lerem os bons textos e os grandes autores; os engenheiros de software podem estudar e avaliar códigos de seu interesse, bem como estabelecer algum tipo de comparação técnica entre os projetos através de métricas de código-fonte.

No Mezuro, as funcionalidades do Kalibro são disponibilizadas em uma rede social para avaliação e monitoramento de código-fonte de projeto de software livre. Nessa rede, pessoas e comunidades podem possuir conteúdos, como blogs, fórums, eventos etc. Entre os tipos de conteúdo, temos no Mezuro os tipos *Kalibro Project* e *Kalibro Configuration*, como ilustrado na Figura 9. O primeiro é para um usuário comum, que deseja

[5] `spago4q.org`

monitorar o código de um projeto e vai usar uma configuração cadastrada no Mezuro. Essa configuração por ser definida por um usuário especialista em métricas que, através do segundo tipo de conteúdo, pode definir sua configuração e compartilhá-la na rede.

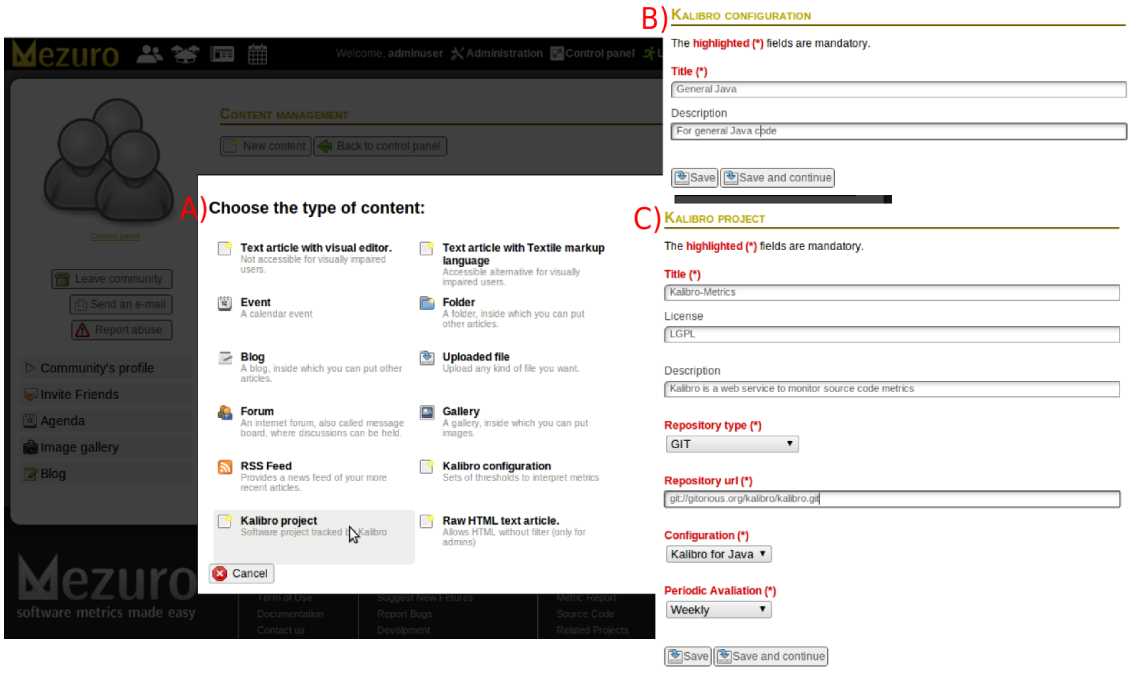

Figura 9. Tipos de conteúdo da rede Mezuro.

O Mezuro é uma extensão da plataforma de rede social e economia solidária Noosfero, escrita em Ruby on Rails. É através de sua plataforma de plugins que os novos tipos de conteúdo são adicionados, entre outras funcionalidades. O plugin do Mezuro se conecta ao Kalibro Service utilizando a *gem* Savon[6], um cliente SOAP para Ruby. Um protótipo do Mezuro pode ser testado em `http://mezuro.org`.

A integração com outros coletores de métricas também está sendo realizada. Atualmente, Kalibro interage com a Analizo, suportando a análise de projetos escritos em C, C++ e Java. O CheckStyle também foi integrado ao Kalibro, e está em fase de testes. Há um esforço paralelo na integração com o CVSAnalY[7], especialmente para coleta de métricas em Python. Por fim, foi iniciada a adaptação da ferramenta JaBUTi[8] para que também sejam disponibilizadas algumas métricas de teste estrutural no Kalibro e no Mezuro.

5. Considerações finais

Neste trabalho foi apresentado o Kalibro Metrics, um serviço Web altamente flexível para a análise de métricas de código-fonte. Em suma, o Kalibro Metrics tem funcionalidades úteis tanto para pesquisadores em engenharia de software que trabalham com análise de código-fonte como para profissionais que querem identificar problemas em potencial e

[6] `savonrb.com`
[7] `projects.libresoft.es/projects/cvsanaly/wiki`
[8] `ccsl.icmc.usp.br/pt-br/projects/jabuti`

possíveis melhoramentos em seu software. Desse modo, ele torna viável o desenvolvimento do Mezuro, um ambiente no qual engenheiros de software podem definir e compartilhar suas próprias configurações de intervalos, de acordo com sua experiência em desenvolvimento de software.

Como uma limitação deste trabalho, pode-se notar que a abordagem aposta em intervalos para auxiliar na interpretação de valores de métricas de código-fonte, cuja utilidade e formas de uso ainda são pontos sob discussão. Espera-se que este trabalho sirva também para aprofundar essa discussão. Muito ainda está para ser discutido a respeito do uso de redes sociais para geração de conhecimento sobre métricas. Entretanto, este não é o foco deste trabalho.

No escopo de trabalhos futuros, estão algumas funcionalidades e otimizações, como a notificação do usuário quando uma análise de projeto cadastrado for finalizada, e a redução do tamanho das requisições, deixando os clientes mais magros, o que vai impactar na melhoria de desempenho da comunicação com o Mezuro.

Referências

Baldwin, C. Y. and Clark, K. B. (2006). The architecture of participation: Does code architecture mitigate free riding in the open source development model? *Management Science*, 52(7):1116–1127.

Beck, K. (1999). *Extreme Programming Explained*. Addison Wesley.

Beck, K. (2007). *Implementation Pattens*. Addison Wesley.

Biswas, R. and Ort, E. (2006). The java persistence api - a simpler programming model for entity persistence. Web Site.

Fenton, N. E. and Neil, M. (1999). Software metrics: successes, failures and new directions. *Journal of Systems and Software*, 47(2-3):149 – 157.

FSF (2011). The free software definition. Free Software Foundation Web Site.

Gamma, E., Johnson, R., Helm, R., and Vlissides, J. M. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Martin, R. C. (2008). *Clean Code - A Handbook of Agile Software Craftsmanship*. Prentice Hall.

Meirelles, P., Santos, C., Miranda, J., Kon, F., Terceiro, A., and Chavez, C. (2010). A study of the relationships between source code metrics and attractiveness in free software projects. *Brazilian Symposium on Software Engineering*, 0:11–20.

Terceiro, A., Costa, J., Miranda, J., Meirelles, P., Rios, L. R., Almeida, L., Chavez, C., and Kon, F. (2010). Analizo: an extensible multi-language source code analysis and visualization toolkit. In *Brazilian Conference on Software: Theory and Practice (CBSoft) – Tools*, Salvador-Brazil.

Vandoren, E. (1997). Maintainability Index Technique for Measuring Program Maintainability. http://www.sei.cmu.edu/str/descriptions/mitmpm.html.

Walton, L. (2011). Metrics. Web Site.