Acompanhamento da Evolução de Software via Métricas

Wallace Ribeiro, Daniel Castellani, Alexandre Plastino, Leonardo Murta

Instituto de Computação – Universidade Federal Fluminense (UFF)

{wribeiro,dribeiro,plastino,leomurta}@ic.uff.br

Resumo. *A análise do histórico de um projeto de software é de suma importância para a sua manutenção, seja para compreender o passado ou prever situações indesejáveis futuras. Contudo, as abordagens atuais para esse problema atuam somente sobre versões completas do software, sem acompanhar revisão à revisão a sua evolução. Dessa forma, o propósito deste trabalho é apresentar mecanismos de apoio à medição de software, viabilizando o acompanhamento da sua evolução em granularidade fina. Esses mecanismos foram aplicados em um sistema de médio porte e foram identificadas correlações entre métricas e razões por trás dessas correlações.*

1. Introdução

No cenário de desenvolvimento de software, a gerência de configuração [1], que monitora o que mudou, onde mudou e quais as consequências das modificações no software, pode ser de grande valia no acompanhamento da sua qualidade, se aplicada em conjunto com métricas de software. Em um projeto de software hospedado em um repositório de controle de versões [2], os programadores inserem suas modificações no código fonte do projeto em diversos momentos. A cada momento em que um projeto de software é modificado, é criada uma nova revisão do software. A coleta de métricas sobre cada uma dessas revisões pode fornecer uma percepção quantitativa do grau em que o produto se encontra em relação a um determinado atributo [1].

Diversos trabalhos já foram desenvolvidos no intuito de coletar e estudar métricas de software. As ferramentas jaBUTi [3], Eclipse Metrics [4] e Analizo [5] são capazes de coletar métricas de software. Em complemento a esses coletores, existem trabalhos cujo objetivo é facilitar a análise das métricas. O Crab [6] e o Kalibro [7], por exemplo, permitem a associação de valores qualitativos a intervalos de valores numéricos das métricas. Além disso, o Crab possibilita a definição de métricas compostas a partir de outras métricas e o Kalibro permite anotar comentários e recomendações aos valores, facilitando dessa maneira a interpretação das métricas. Contudo, além dessas abordagens atuarem de forma desintegrada, elas exigem o cadastro prévio dos intervalos de valores de análise e possibilitam somente a análise do produto como um todo.

Por outro lado, o trabalho proposto neste artigo tem como objetivo principal não somente a análise integrada das características do produto como um todo, mas também de cada modificação individual feita durante o seu ciclo de vida. Dessa forma, essa proposta permite o estudo das modificações ocorridas em diferentes revisões de um software, tendo como foco as modificações dos valores de suas métricas. Para tal, foram

elaborados mecanismos que extraem métricas, chamadas de métricas básicas, pois são extraídas diretamente do produto [8]. Além disso, para viabilizar análises mais complexas, que correlacionem diferentes métricas, foi proposto um mecanismo que, utilizando as métricas previamente definidas, permite a criação de métricas derivadas (ou compostas). Com a capacidade de coleta de métricas básicas e derivadas, um terceiro mecanismo foi desenvolvido para exibir graficamente os valores das métricas e suas modificações com o passar do tempo.

Este artigo está organizado em seis seções além desta introdução. Na Seção 2, são apresentadas as métricas básicas implementadas neste trabalho. Na Seção 3, é apresentado o recurso de definição de métricas derivadas a partir de outras métricas. Na Seção 4, são detalhados os gráficos de visualização das métricas no decorrer do tempo. Na Seção 5, os mecanismos propostos são aplicados sobre um sistema real, a fim de avaliar as possibilidades de investigação propiciadas por este trabalho. Por fim, a Seção 6 apresenta as contribuições deste trabalho e aponta alguns trabalhos futuros.

2. Métricas Básicas

Os mecanismos de coleta de métricas desenvolvidos neste trabalho são capazes de lidar atualmente com 22 métricas de software. Essas métricas são extraídas dos produtos de software e seus valores são armazenados em um banco de dados. A **Tabela 1** mostra as métricas implementadas e algumas de suas características: nome da métrica, sua sigla, uma descrição em alto nível e o domínio de seus valores.

Tabela 1. Métricas básicas implementadas

NOME	SIGLA	DESCRIÇÃO	DOM.
Abstractness	RMA	Indica a razão entre as classes abstratas e o número total de classes de um pacote.	[0,1]
Average Number Of Ancestors	ANA	Indica o número médio de classes das quais cada classe do projeto herda informações.	[1,∞]
Class Interface Size	CIS	Indica o número de métodos públicos em uma classe.	[0,∞]
Cohesion Among Methods in Class	CAM	Indica a coesão entre os métodos de uma classe.	[0,1]
Cyclomatic Complexity	MCC	Indica o número de caminhos independentes que o programa pode tomar durante a execução.	[1,∞]
Data Access Metric	DAM	Indica a razão entre os atributos privados (e protegidos) e o número total de atributos.	[0,1]
Design Size in Classes	DSC	Indica o número de classes de um projeto.	[0,∞]
Direct Class Coupling	DCC	Indica o número de classes diferentes com que uma classe se relaciona.	[0,∞]
Lack Of Cohesion Of Methods	LCOM	Indica quanto os métodos de uma classe se relacionam.	[0,∞]
Lines Of Code	LOC	Indica o somatório de linhas de código de uma classe, incluindo comentários.	[0,∞]
Measure Of Aggregation	MOA	Indica o número de declarações de dados, cujos dados são definidos pelo usuário.	[0,∞]

NOME	SIGLA	DESCRIÇÃO	DOM.
Measure Of Functional Abstraction	MFA	Indica a razão entre os métodos herdados e todos os métodos acessíveis de uma classe.	[0,1]
Methods Lines Of Code	MLOC	Indica o número de linhas de código por método.	[0,∞]
Number Of Attributes	NOA	Indica o número de atributos de uma classe.	[0,∞]
Number Of Hierarchies	NOH	Indica o número total de hierarquias de um projeto.	[0,∞]
Number Of Interfaces	NOI	Indica o número de interfaces de um pacote.	[0,∞]
Number Of Methods	NOM	Indica o número de métodos de uma classe.	[0,∞]
Number of Overridden Methods	NORM	Indica o número de métodos sobrescritos de um projeto. Métrica de projeto	[0,∞]
Number of Polymorphic Methods	NOP	Indica o número de métodos que apresentam comportamento polimórfico.	[0,∞]
Number Of Static Attributes	NSF	Indica o número de atributos estáticos de uma classe.	[0,∞]
Number Of Static Methods	NSM	Indica o número de métodos estáticos de uma classe.	[0,∞]
Total Cyclomatic Complexity	TCC	Indica a soma da complexidade ciclomática de todas os métodos de uma classe.	[0,∞]

Para que as métricas possam ser extraídas, é necessário que produto de software esteja escrito na linguagem de programação Java [9], que seja gerenciado pelo Maven [10] e que esteja hospedado em um repositório Subversion [11]. Essas restrições existem visto que o mecanismo de coleta de métricas é completamente automatizado.

Para implementar a coleta das métricas foram utilizados algumas bibliotecas licenciadas como software livre: JDepend [12], Dependency Finder [13], Byte Code Engineering Library (BCEL)[14], CKJM [15], JavaNCSS [16] e Locc [17].

3. Métricas Derivadas

Com as métricas básicas, descritas na Seção 2, pode-se avaliar quantitativamente características sobre projetos de software. Porém, por avaliarem características de baixo nível, como quantidade de métodos ou acoplamento, a avaliação da qualidade do produto com base nelas fica comprometida.

Dessa forma, para que sejam avaliadas características de qualidade de alto nível, foi criado um mecanismo de definição de métricas compostas. Assim, pode-se combinar métricas de forma a construir métricas de mais alto nível. Para se criar métricas compostas, foi elaborado um mecanismo que permitisse montar expressões matemáticas de métricas com o objetivo de formar novas métricas. As métricas compostas podem utilizar métricas básicas ou outras métricas compostas nessas expressões. Desse modo, tem-se um sistema flexível para criação de novas métricas a partir de outras.

Por exemplo, à medida que o projeto cresce, LOC aumenta e TCC também. Assim, a métrica derivada Complexidade Média por Método (CMM) é útil quando se deseja identificar a complexidade média dos métodos, pois ela mostra a razão entre TCC e NOM e não sofre influência do crescimento do projeto.

4. Monitoramento Gráfico

Um monitor gráfico é uma ferramenta de grande importância para o estudo das métricas. Sabendo-se que serão coletadas muitas métricas e que um projeto de software usualmente tem em torno de algumas centenas ou milhares de revisões, a quantidade de informação pode ser grande demais para que se possa tirar alguma conclusão sobre esses dados com o auxílio de tabelas de números apenas. Gráficos podem condensar o grande volume de informação e mostrá-la de uma forma mais intuitiva.

Por esse motivo criou-se um conjunto de gráficos que pudesse facilitar o exame e estudo das métricas. Foram escolhidos, para tanto, gráficos de controle e histogramas. Um gráfico de controle é um gráfico que possui uma linha de limite superior, uma linha de limite inferior e uma linha central que, geralmente, mostra a média dos valores do conjunto de amostras. Já um histograma é um gráfico, normalmente de barras verticais, que representa a distribuição de frequências do conjunto de amostras.

Neste projeto, o conjunto de amostras é formado pelos valores das métricas das revisões de um software. Cada gráfico utiliza como base os valores de uma determinada métrica de um projeto específico em suas diversas revisões. O gráfico de controle mostra uma linha central representando a média desses valores, duas linhas (uma superior e outra inferior) representando o limite de um desvio padrão para mais e para menos, e duas linhas (uma superior e outra inferior) representando o limite de três desvios padrão para mais e para menos. Gráficos de controle costumam também possuir duas linhas representando o limite de dois desvios padrão (para mais e para menos), porém, neste trabalho, considerou-se não adicionar essas linhas por medida de simplificação. Já o histograma mostra a distribuição das frequências desses valores.

Para a geração de gráficos, foi utilizada a biblioteca JfreeChart [18], que, uma vez tendo um conjunto de dados organizados, consegue criar gráficos de forma programática. O ambiente foi desenvolvido como uma aplicação web. Em função disso, uma vez tendo os gráficos, foi possível exibi-los utilizando a tecnologia JSF.

5. Avaliação Experimental

Para avaliar as funcionalidades desta proposta, foi escolhido um projeto de software, extraídas algumas de suas métricas e estudados alguns de seus gráficos. O projeto escolhido foi o Publico Core, um módulo do sistema IdUFF – sistema de gestão acadêmica da Universidade Federal Fluminense. O Publico Core é o sistema que auxilia o IdUFF no acesso a dados compartilhados por outros sistemas, como dados de alunos e funcionários. Escolheu-se o Publico Core por ser um sistema relativamente simples, porém de grande importância para vários sistemas da UFF. As métricas extraídas foram: *Number Of Methods*, *Total Cyclomatic Complexity* e *Lines Of Code*. O objetivo foi estudar o comportamento do projeto com relação às métricas escolhidas, observar as variações de cada métrica e tentar inferir algumas implicações para o comportamento observado. Essas métricas foram escolhidas, pois, neste estudo, tenta-se encontrar as relações entre a quantidade de código escrito e complexidade total do sistema.

No estudo do gráfico da métrica *Number Of Methods*, exibido na Figura 1, verifica-se que o número de métodos do projeto tende a crescer de forma contínua e pouco acentuada em grande parte das revisões. Entretanto, o primeiro ponto que chama

a atenção ocorre entre a décima e décima segunda revisões. A décima revisão possui 399 métodos, enquanto que a décima segunda revisão possui 653 métodos, significando um aumento de 254 métodos. Também, observa-se outro trecho com aumento relativamente significativo, entre a décima quinta (653 métodos) e décima sexta (752 métodos) revisões. Outro ponto interessante é que o único trecho onde o número de métodos diminui é entre a vigésima e vigésima primeira revisões, tendo respectivamente, 774 e 735 métodos. Após esses pontos, o gráfico tende a ter um aumento pequeno e contínuo de métodos.

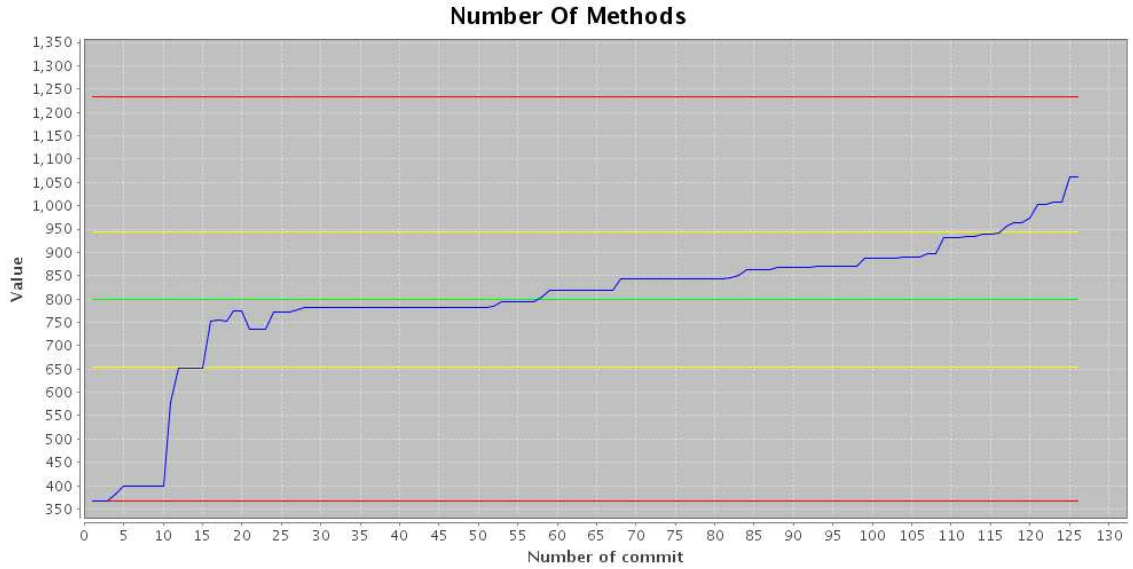

Figura 1. Gráfico do valor absoluto da métrica *Number Of Methods*.

Investiga-se então se existe alguma relação entre os valores da métrica *Number Of Methods* e os valores das métricas *Total Cyclomatic Complexity* e *Lines Of Code*, estudando principalmente as revisões que chamaram atenção na métrica *Number Of Methods*. Primeiro, verifica-se o gráfico da métrica *Lines Of Code*, exibido na Figura 2. Nessa métrica, a décima revisão possui valor 7.199, enquanto que a décima segunda revisão possui valor 7.205, ou seja, houve um aumento de 6 linhas de código entre as revisões. Também é verificado que a décima quinta revisão possui 7.209 linhas de código e a décima sexta revisão possui 7.693 linhas, um aumento de 484 linhas. Observa-se que, na vigésima revisão, o projeto possui 7.879 linhas e, na vigésima primeira revisão, possui 7.579 linhas, uma diminuição de 300 linhas.

Por último, estuda-se o comportamento da métrica *Total Cyclomatic Complexity*, cujo gráfico encontra-se na Figura 3. Observa-se que essa métrica possui um aumento constante e pouco acentuado de seus valores. Descobre-se que a décima revisão possui complexidade total de 1.184 e a décima segunda revisão, de 1.173, uma diminuição de 11 no valor da métrica. A décima quinta revisão possui 1.173 e a décima sexta revisão possui 1.221. Na vigésima revisão, o projeto possui complexidade ciclomática total de 1.245, já na vigésima primeira revisão possui complexidade de 1.203.

Comparando as três métricas, alguns fatos interessantes podem ser observados. O primeiro deles é entre a décima e décima segunda revisões. Nesse trecho, há um aumento de 254 métodos, 6 linhas de código e diminuição de 11 de complexidade ciclomática total. Com uma diferença tão pequena no número de linhas de código, pode-

se inferir que o que houve na verdade foi uma refatoração, que pretendia separar funcionalidades em mais métodos. Nesse caso, a complexidade ciclomática total não deveria aumentar muito, já que não houve adição de funcionalidades, apenas o espalhamento dessas funcionalidades em mais métodos. O que se observou foi que, na verdade, houve uma diminuição da complexidade ciclomática total. Entre a décima quinta e décima sexta revisões, há um aumento de 99 métodos, 484 linhas e 48 na complexidade ciclomática. Esse trecho chama atenção inicialmente pelo grande aumento no número de métodos em relação aos outros trechos do gráfico. Entretanto, esse trecho se comporta dentro da normalidade de um período de criação de funcionalidades, onde o número de métodos aumenta, o número de linhas aumenta e a complexidade total aumenta. Ainda é possível observar que, entre a vigésima e vigésima primeira revisões, há uma diminuição de 39 métodos, 300 linhas de código e 42 de complexidade ciclomática. Talvez, nesse caso, tenha havido a retirada de alguma funcionalidade ou substituição por uma mais simples.

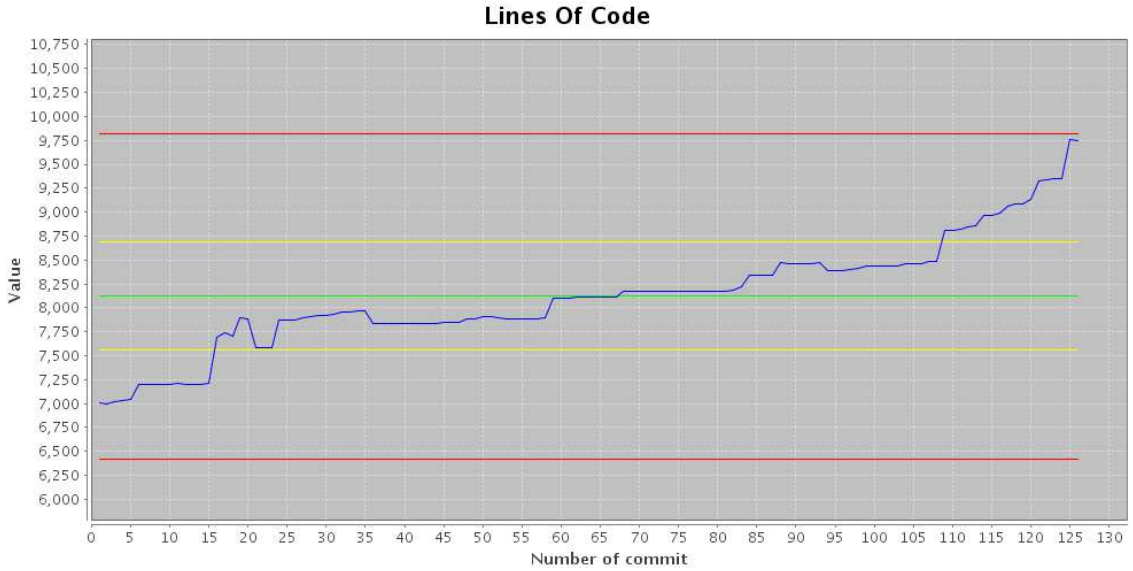

Figura 2. Gráfico do valor absoluto da métrica *Lines Of Code*.

Após esses trechos, os gráficos tendem a ter um comportamento mais estável, o que pode significar que o sistema encontra-se em um ritmo estável de desenvolvimento. Entretanto, pode-se observar, no gráfico da métrica *Total Cyclomatic Complexity*, que, entre a penúltima (revisão 21511) e última (revisão 21633) revisões, existe um decréscimo de valor. Na revisão 21511, o valor da métrica é 1.580 e, na revisão 21633, o valor é 1.570, uma diminuição de 10. Observando o gráfico de *Number of Metohods*, a penúltima revisão possui o valor de 1.061 e a última revisão possui o valor de 1.062, ou seja, houve o aumento de 1 método. Também há um decréscimo de 6 linhas de código, sendo 9.756 linhas na penúltima revisão e 9.750 linhas na última revisão. Um aumento do número de métodos seguido pelo não aumento do número de linhas pode ser um forte indício de uma refatoração. Nesse caso, ainda houve um decréscimo do número de linhas, o que quer dizer que além de dividir a funcionalidade em mais métodos, cada método se tornou suficientemente pequeno para que a soma de suas linhas pudesse ser menor que a do método original. Consequentemente, essa diminuição de linhas pode gerar uma diminuição de complexidade e foi, na realidade, o que ocorreu.

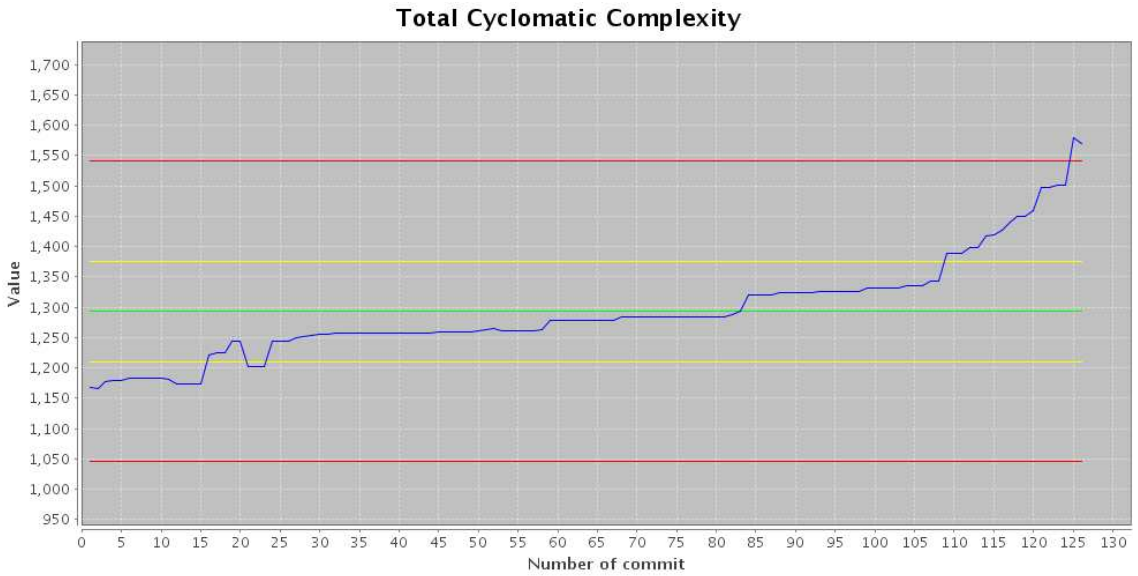

Figura 3. Gráfico do valor absoluto da métrica *Total Cyclomatic Complexity*.

6. Conclusão

Este trabalho reúne uma gama de métricas que podem ser utilizadas diretamente ou adaptadas para serem utilizadas em outros sistemas. Uma vez possuindo um projeto escrito em Java, gerenciado pelo Maven e hospedado em um repositório Subversion, o sistema proposto extrai as métricas de uma forma transparente ao usuário. Outra contribuição foi permitir a criação de métricas através de outras métricas. Esse ponto tem significante importância, principalmente para usuários que desejam testar e estudar seus próprios projetos em profundidade.

Por último, foram criados os gráficos que fornecem uma forma integrada de interação com os valores coletados pelas métricas. Cada projeto pode possuir centenas ou milhares de revisões. Um número tão grande de revisões gera um número enorme de dados, cujo estudo poderia ser extremamente demorado e contraproducente. Uma forma de se resolver o problema da grande quantidade de dados constituiu em criar um gráfico de controle que pudesse mostrar, em uma figura, a variação dos valores das métricas em relação ao tempo decorrido.

Como trabalhos futuros vislumbra-se a adoção de mecanismos de filtragem de versões para possibilitar análises espaçadas ou parciais da história do projeto. Além disso, outro trabalho futuro importante seria permitir que diferentes usuários adicionassem anotações ao gráfico, além de anotações automáticas que poderiam ser produzidas junto com os gráficos (anotações de marcos do projeto, por exemplo). Essas anotações possibilitariam auditorias conjuntas sobre a história do projeto. Por fim, estudos experimentais mais detalhados contribuiriam para identificar os potenciais dessa tecnologia.

7. Agradecimentos

Gostaríamos de agradecer ao CNPq (processos 307078/2009-4 e 305283/2011-1) e FAPERJ (processos E-26/103.087/2008, E-26/111.281/2011 e E-26/103.253/2011) pelo apoio financeiro.

References

[1] R. S. Pressman, Software Engineering: A Practitioner's Approach, vol. 6th. McGraw-Hill, 2005.

[2] S. Pherigo, T. Mikkelsen, and S. Pherigo, Practical Software Configuration Management: The Latenight Developer's Handbook. Prentice Hall PTR, 1997.

[3] "jabutimetrics - Projeto JaBUTi Metrics: Implementação de Avaliação Automática de Métricas de Orientação a Objetos na JaBUTi - Google Project Hosting". [Online]. Available: http://code.google.com/p/jabutimetrics/. [Accessed: 04-maio-2012].

[4] "Metrics 1.3.6". [Online]. Available: http://metrics.sourceforge.net/. [Accessed: 04-maio-2012].

[5] "Analizo - Web Home". [Online]. Available: http://analizo.org/. [Accessed: 04-maio-2012].

[6] "Crab_PauloMeirelles_artigoSBES2009_Final.pdf (objeto application/pdf)". .

[7] "Kalibro Metrics | CCSL". [Online]. Available: http://ccsl.ime.usp.br/kalibro. [Accessed: 04-maio-2012].

[8] ISO, "ISO/IEC 9126 - Software engineering - Product quality," International Organization for Standardization, 2001.

[9] J. Gosling, B. Joy, G. Steele, and G. Bracha, The Java Language Specification, vol. 3rd. Addison-Wesley Professional, 2005.

[10] Jason Van Zyl, Maven: Definitive Guide, First. [[O'Reilly Media]], 2008.

[11] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato, Version Control with Subversion, vol. 2nd. .: O'Reilly Media, 2008.

[12] Clarkware Consulting, Inc, "JDepend," 11-Mar-2011. [Online]. Available: http://clarkware.com/software/JDepend.html. [Accessed: 11-Mar-2012].

[13] Jean Tessier, "Dependency Finder." [Online]. Available: http://depfind.sourceforge.net/. [Accessed: 11-Mar-2012].

[14] Apache Software Foundation, "Apache Commons BCELTM -," 11-Mar-2011. [Online]. Available: http://commons.apache.org/bcel/. [Accessed: 11-Mar-2012].

[15] K. Chidamber, "ckjm — Chidamber and Kemerer Java Metrics," 11-Mar-2011. [Online]. Available: http://www.spinellis.gr/sw/ckjm/. [Accessed: 11-Mar-2012].

[16] Chr. Clemens Lee, "JavaNCSS - A Source Measurement Suite for Java," 11-Mar-2011. [Online]. Available: http://javancss.codehaus.org/. [Accessed: 11-Mar-2012].

[17] Collaborative Software Development Laboratory, "LOCC — Collaborative Software Development Laboratory," 11-Mar-2011. [Online]. Available: http://csdl.ics.hawaii.edu/Plone/research/locc/. [Accessed: 11-Mar-2012].

[18] D. Gilbert and T. Morgner, "JFreeChart, a free Java class library for generating charts," 2011. [Online]. Available: http://www.jfree.org/jfreechart. [Accessed: 10-Oct-2011].