# Graphical and Statistical Analysis of the Software Evolution Using Coupling and Cohesion Metrics - An Exploratory Study

Raul Silva
University of Norte do Paraná
Londrina
Paraná - Brasil
raulsfe@gmail.com

Heitor Costa
Department of Computer Science
Federal University of Lavras
Lavras, Brazil
heitor@dcc.ufla.br

***Abstract*—Developing software is expensive; thus keep it useful to its users is important. On the other hand, due to constant maintenance performed to meet the changing needs of users, software undergoes degradation of its internal structure, particularly in coupling and cohesion. Monitoring the development of software by using some of its versions can aid Software Engineer with relevant information to guide your maintenance activities. In this paper, we presented a view of the evolution of versions of software. For this, a study was conducted in 10 versions of FindBugs using coupling and cohesion metrics calculated from VizzMaintenance and Metric plug-ins. In this study, we applied the Pearson linear correlation analysis among measurements. The result showed that there is some correlation between these metrics, because coupling metrics directly influenced the cohesion metrics, with undesirable characteristics such as high coupling and low cohesion compromising software quality.**

***Keywords*—Software Evolution; Software Quality; Coupling and Cohesion Metrics**

## I. INTRODUÇÃO

A evolução de sistemas de software consiste em realizar mudanças tendo em vista novos requisitos, gerados por alterações em regras de negócio ou para corrigir erros que impedem o funcionamento normal desses sistemas [12, 13]. Essa evolução é estudada a mais de 20 anos e teve seu início na década de 60 quando foram formuladas oito leis para evolução, amplamente aceitas. Porém, evoluir sistemas de software de maneira rápida e eficiente é um dos maiores desafios na engenharia de software [17]. Principais indicadores que um sistema de software tem a necessidade de evoluir são [9] código complexo, histórico de mudanças frequentes, mudanças dispersas no código e soluções temporárias. Outros fatores não diretamente ligados a escrita do código, mas podem indicar necessidade de evolução são, por exemplo,

- Arquitetura inadequada;
- Violação dos princípios de *design* originais;
- Requisitos imprecisos;
- Pressão de tempo para desenvolver o produto;
- Ferramentas de programação inadequadas;
- Ambiente organizacional não adequado;
- Diferentes níveis de programadores;
- Processo de mudanças inadequado.

Um problema inerente a essa evolução diz respeito à carência de técnicas e de ferramentas que auxiliam no controle do processo evolutivo [21]. Analisar essa evolução pode ajudar na classificação e na verificação de um sistema de software, a qual pode ser realizada utilizando medidas de software. Essas medidas avaliam um atributo (propriedade ou características) de uma entidade (produto, processo ou recursos) que podem ser classificadas de várias formas, por exemplo, medidas de tamanho, de acoplamento, de coesão, de ocultação de informações, de herança, de polimorfismo e de reúso. São exemplos de medidas:

- Tamanho de sistemas de software (por exemplo, quantidade de linhas de código);
- Quantidade de pessoas necessárias para implementar um caso de uso;
- Quantidade de defeitos encontrados por fase de desenvolvimento;
- Esforço para realizar uma tarefa;
- Tempo para realizar uma tarefa;
- Custo para realizar uma tarefa;
- Grau de satisfação do cliente.

Classes, funções, métodos e variáveis devem ser medidas em um *design* orientado a objetos com medidas de tamanho, de qualidade e de complexidade para gerar informações relevantes do projeto e averiguar quais partes do software precisam ser otimizadas ou consertadas [11]. Vários conjuntos de medidas podem ser utilizados, por exemplo, suíte CK [7], MOOD *(Metrics for Object-Oriented Design*) [1] e QMOOD (*Quality Metrics for Object-Oriented Design*) [2].

Neste trabalho, o objetivo foi realizar um estudo exploratório em um sistema de software de código aberto desenvolvido na linguagem de programação Java para verificar

978-1-4673-9143-6/15/$31.00 ©2015 IEEE

como foi, historicamente, sua evolução com relação ao impacto nas propriedades de acoplamento e de coesão de suas classes. Para isso, foram analisadas 10 versões consecutivas do software FindBugs, utilizando medidas de acoplamento e de coesão. O valor dessas medidas foi obtido a partir dos *plug-ins* VizzMaintenance[1] e Metric[2]. Em seguida, para verificar a existência de relacionamento entre as medidas utilizadas, foi realizada a análise de correlação linear de Pearson entre essas medidas.

Os resultados apresentados mostraram que há deterioração do nível de acoplamento (alto) e de coesão (baixo) entre algumas versões do sistema de software FindBugs. Além disso, foi constatada relação entre algumas medidas de acoplamento e de coesão quanto a correlação de Pearson foi utilizada. Assim, pode-se concluir que, aparentemente, houve pouca preocupação com as duas propriedades analisadas (acoplamento e coesão), bem como há alguma relação entre medidas de acoplamento e medidas de coesão.

O restante do artigo está organizado da seguinte forma. As medidas utilizadas e a técnica estatística realizada são brevemente apresentadas na Seção II. A metodologia utilizada para a realização deste trabalho é abordada na Seção III. A análise da evolução do software estudado é discutida na Seção IV. Alguns trabalhos relacionados estão resumidos na Seção V. Conclusões, contribuições e sugestões de trabalhos futuros estão descritos na Seção VI.

## II. Referencial Teórico

### A. Medidas Utilizadas

Neste estudo, algumas medidas foram utilizadas para analisar a evolução do software `FindBugs` utilizando medidas de acoplamento e de coesão. Foram utilizadas sete medidas, sendo duas medidas de coesão e cinco medidas de acoplamento (Table I).

TABLE I. Medidas Utilizadas para Realizar Análise da Evolução do Sistema de Software FindBugs

| # | Medidas | Tipos de Medidas |
|---|---|---|
| 1 | LCOM (*Lack of Cohesion in Methods*) | Coesão |
| 2 | TCC (*Tight Class Cohesion*) | Coesão |
| 3 | CBO (*Coupling Between Object*) | Acoplamento |
| 4 | DAC (*Data Abstraction Coupling*) | Acoplamento |
| 5 | MPC (*Message Passing Coupling*) | Acoplamento |
| 6 | Ce (*Efferent Coupling*) | Acoplamento |
| 7 | Ca (*Afferent Coupling*) | Acoplamento |

As medidas LCOM, CBO, DAC e MPC estão definidas na suíte CK [7], amplamente utilizada na literatura. As medidas Ce, Ca e TCC verificam propriedades importantes para um sistema orientado a objetos. Com as medidas Ca e Ce, são obtidos valores que representam o grau de acoplamento entre os pacotes de um sistema de software orientado a objetos. Com a medida TCC, o valor obtido corresponde ao grau de coesão dos métodos de uma classe. Um fator relevante para a escolha dessas medidas é a existência de ferramentas computacionais, (VizzMaintenance e Metric) que calculam o seu valor.

Com a medida LCOM (*Lack of Cohesion Methods*), pode-se medir a desigualdade de métodos em uma classe, tendo em vista a variável de instância ou atributos utilizados por métodos [7]. Considerando uma classe $C_1$ com n métodos $M_1$, $M_2$, ..., $M_n$. Seja $I_j$ o conjunto de variáveis de instância utilizadas pelo método $M_i$. Existem n conjuntos tais como $\{I_1\}$, ..., $\{I_n\}$. Seja

$$P = \{(I_i, I_j) / I_i \cap I_j = 0\}$$

$$Q = \{(I_i, I_j) / I_i \cap I_j \neq 0\}$$

Se os n conjuntos $\{I_1, ..., I_n\}$ são iguais a 0, então P = 0.

$$\mathrm{LCOM} = \begin{cases} |P| - |Q|, & \text{se } |P| > |Q| \\ 0, & \text{caso contrário} \end{cases}$$

Na medida TCC, é considerada uma classe C, na qual NP(C) é a quantidade máxima de pares de métodos da classe

$$\mathrm{NP(C)} = \frac{n * (n-1)}{2}$$

A quantidade de pares de métodos (m, n) que possuem conexão direta na classe é representada por NDC(C).

NDC(C) = |{(m, n) / m e n acessam um variável de instância em comum}|

Um método está diretamente conectado a outro se ambos utilizam pelo menos uma variável de instância em comum da classe. TCC é dada por

$$\mathrm{TCC} = \frac{\mathrm{NDC(C)}}{\mathrm{NP(C)}}$$

sendo um percentual de pares de métodos com conexão direta na classe. Os resultados de TCC são normalizados no intervalo [0, 1], sendo o valor 1 correspondente a alta coesão.

Com a medida CBO (*Coupling Between Object*), pode-se obter o grau de acoplamento entre as classes de um sistema de software. Quanto maior a ligação entre elas, menor a possibilidade de reúso, pois a classe torna-se dependente de outras classes para cumprir suas obrigações. Portanto, a medida CBO está diretamente ligada ao nível de reaproveitamento. Alto acoplamento indica baixa independência de classe, o que aumenta a complexidade e o esforço de teste (Fig. 1) [7, 5].

A medida DAC (*Data Abstraction Coupling*) é definida como a complexidade causada por tipos abstratos de dados (TAD) [19]. Essa medida está relacionada ao acoplamento

[1] https://marketplace.eclipse.org/content/vizzmaintenance
[2] http://marketplace.eclipse.org/content/eclipse-metrics

entre as classes que representam um aspecto importante do projeto orientado a objetos, uma vez que o grau de reúso, o esforço de manutenção e os testes para uma classe são decisivamente influenciados pelo nível de acoplamento entre classes:

DAC = quantidade de TADs definidos em uma classe

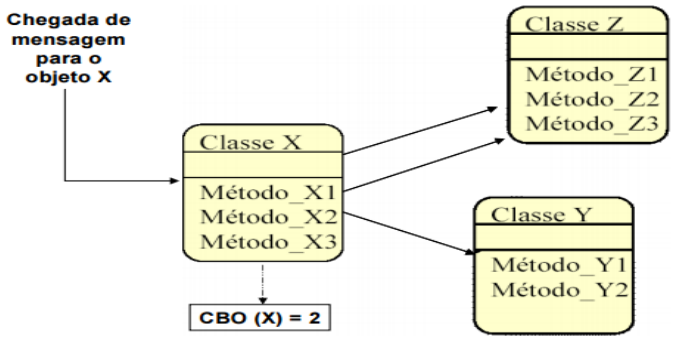


Fig. 1. Exemplo de Valor da Medida CBO

A medida MPC (*Message Passing Coupling*) é definida como a quantidade de chamadas de métodos a outras classes [19]. Portanto, se dois métodos diferentes na classe A acessam o mesmo método na classe B, então MPC(A) = 2. Considerando o Diagrama de Classes apresentado da Fig. 1, MPC(X) = 3.

A medida Ca (*Afferent Coupling*) é destinada a pacotes e corresponde à quantidade de classes em outros pacotes que dependem de classes dentro do pacote [15]. A medida Ce (*Efferent Coupling*) é destinada a pacotes e corresponde à quantidade de classes em outros pacotes que as classes do pacote depende [15].

### B. *Análise Estatística*

Neste trabalho, foi verificada a existência da relação entre as medidas extraídas, isto é, descobrir se as alterações sofridas por uma medida são acompanhadas estatisticamente por alterações em outras medidas. Para isso, foi utilizada a correlação linear de Pearson. Com a correlação linear de Pearson, pode-se mensurar a intensidade da associação linear existente entre as variáveis com a obtenção do coeficiente de correlação:

$$r = \frac{C_{x,y}}{S_x * S_y}, \quad r \in [-1, 1]$$

sendo $C_{x,y}$ a Covariância ou variância conjunta das variáveis X e Y, $S_x$ o desvio padrão da variável X e $S_y$ o desvio padrão da variável Y.

O valor desse coeficiente varia entre -1 e 1. O valor 0 (zero) significa que não há relação linear, o valor 1 indica relação linear perfeita e o valor -1 indica uma relação linear perfeita, mas inversa, ou seja, quando uma das variáveis aumenta a outra diminui. Quanto mais próximo estiver de 1 ou -1, mais forte é a associação linear entre as duas variáveis.

## III. METODOLOGIA PROPOSTA

A metodologia proposta é apresentada na Fig. 2, sendo constituída por quatro etapas:

- **Definição do Estudo.** O objetivo do estudo é investigar as propriedades de acoplamento e de coesão de um sistema de software. Além disso, verificar se as medidas utilizadas influenciam em função da outra e vice-versa. Um levantamento foi realizado para verificar as medidas de acoplamento e coesão existentes, em seguida foi verificado quais ferramentas realizava a coleta dessas medidas. Ao final foi definido dois programas coletadores e sete medidas, sendo cinco de acoplamento e duas de coesão;
- **Software Analisado**. Alguns sistemas de software foram analisados para verificar se suas últimas versões consecutivas e disponíveis estavam funcionais;
- **Coleta e Análise**. Para acompanhar e analisar a evolução, o valor das medidas sobre as versões do software foi coletado e a evolução foi analisada graficamente. Por fim, esses dados foram inter-relacionados utilizando correlação linear de Pearson e discutidos.

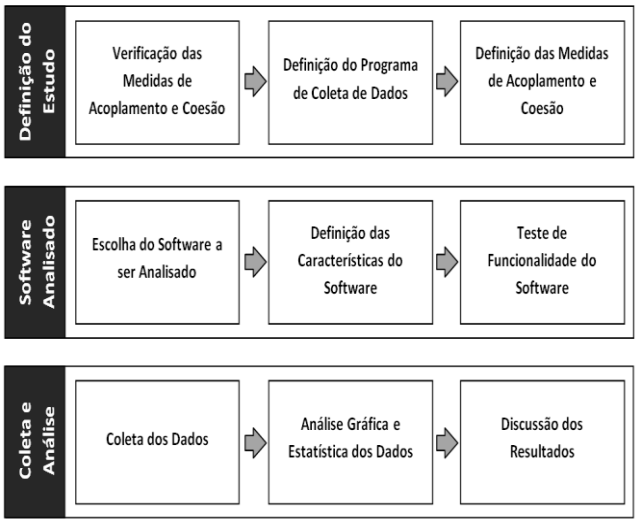


Fig. 2. Metodologia Utilizada

## IV. ANÁLISE DA EVOLUÇÃO DO SOFTWARE

Analisar a evolução de sistemas de software desde o início é importante para verificar possível degradação da qualidade da estrutura interna.

### A. *Características do Software*

O sistema de software analisado foi o `FindBugs`[3], o qual é utilizado para encontrar erros no código Java realizando análise estática. É um software livre, de código fonte aberto, desenvolvido com a linguagem de programação Java, que apresenta características de um sistema de software relativamente grande, o que influenciou na sua escolha. Além disso, ele é distribuído sob os termos da *Lesser GNU Public Licence*. Seu nome e logotipo são marcas registradas da Universidade de Maryland. Esse sistema de software pode ser executado de várias formas, por exemplo, linha de comando e *plug-in* para alguns IDEs (Eclipse[4], NetBeans[5], IntelliJ IDEA[6], Hudson[7] e Jenkins[8]).

Assim, para a realização estudo, foi utilizada a última versão disponível e funcional do software em questão e 9 versões disponíveis, funcionais e consecutivas anteriores, totalizando 10 versões do sistema de software FindBugs. À época da realização deste estudo, a versão do FindBugs era 3.0.0, a qual possui 1.223 classes (*Number of Classes*), 125.893 linhas de código (*Total Line of Code*), 78 pacotes (*Number of Package*) e 8.642 métodos (*Number of Methods*) (Table II).

TABLE II. MEDIDAS UTILIZADAS PARA REALIZAR ANÁLISE DA EVOLUÇÃO DO SISTEMA DE SOFTWARE FINDBUGS

| # | Versão | Quantidade de Classes | Quantidade de Atributos | Total de Linhas de Código | Quantidade de Métodos | Quantidade de Pacotes |
|---|---|---|---|---|---|---|
| 1 | 1.3.5 | 1.089 | 3.073 | 98.764 | 7.446 | 67 |
| 2 | 1.3.6 | 1.087 | 3.081 | 99.685 | 7.479 | 67 |
| 3 | 1.3.7 | 1.088 | 3.099 | 100.282 | 7.513 | 67 |
| 4 | 1.3.8 | 1.101 | 3.148 | 102.601 | 7.619 | 67 |
| 5 | 1.3.9 | 1.162 | 3.374 | 110.782 | 8.101 | 67 |
| 6 | 2.0.0 | 1.189 | 3.374 | 114.632 | 8.469 | 74 |
| 7 | 2.0.1 | 1.194 | 3.398 | 115.495 | 8.522 | 74 |
| 8 | 2.0.2 | 1.197 | 3.402 | 116.100 | 8.551 | 75 |
| 9 | 2.0.3 | 1.212 | 3.433 | 118.104 | 8.595 | 77 |
| 10 | 3.0.0 | 1.223 | 3.468 | 125.893 | 8.642 | 78 |

### B. *Tecnologias Utilizadas*

Com o intuito de analisar a qualidade da estrutura interna do sistema de software FindBugs, foi utilizada a *IDE Eclipse Luna*[9], software livre mantido pelo *Eclipse Foundation*, que utiliza os *plug-ins* VizzMaintenance e Metrics para a obtenção do valor das medidas utilizadas neste trabalho.

Com o *plug-in* VizzMaintenance, pode-se realizar análises em sistemas de software Java, apresentando informações detalhadas sobre classes e quais devem ser reformuladas para melhorar a sua manutenção. Utiliza análise estática para calcular 17 medidas de software. Com o *plug-in* Metrics, pode-se realizar o cálculo de 22 medidas que fornecem informações, tais como, valor total, valor médio, valor máximo e desvio padrão. Os dados coletados por esses *plug-ins* podem ser exportados para, por exemplo, uma planilha eletrônica.

### C. *Análise*

Para a realização da coleta de dados, as 10 últimas versões do sistema de software FindBugs foram importadas para a ferramenta *Eclipse Luna*. Em seguida, o valor das medidas de coesão LCOM e TCC e o valor das medidas de acoplamento CBO, DAC, MPC foram extraídos a partir do *plug-in* VizzMaintenance e o valor das medidas de acoplamento Ce e Ca foram extraídas com o auxílio do *plug-in* Metrics. Para o valor das medidas analisadas, foi elaborado um gráfico representando sua evolução de forma isolada e facilitando a análise de seu comportamento em relação a cada versão.

Após, o valor das medidas de acoplamento foi correlacionado com o valor das medidas de coesão, utilizando a análise de correlação linear de Pearson. Os resultados que apresentaram correlação linear significativa ($p < 0,05$) foram representados graficamente e discutidos. As análises estatísticas foram realizadas utilizando o programa estatístico R [19].

#### 1) *Análise Individual das Medidas*

O valor da medida LCOM para as 10 versões do FindBugs pode ser analisado na Fig. 3. Percebe-se que esse valor aumenta à medida que há a evolução do sistema de software, caracterizando crescente falta de coesão. Além disso, o valor da medida TCC para as 10 versões do FindBugs pode ser analisado na Fig. 4. Percebe-se que esse valor aumenta até a versão 1.3.8, havendo diminuição do valor para as demais versões.

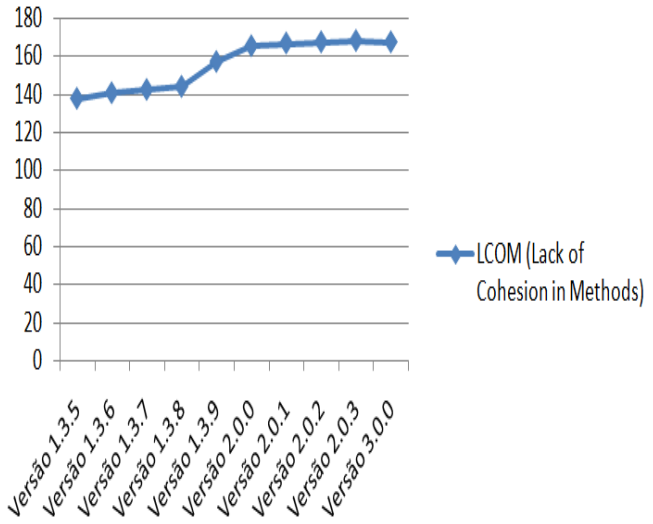


Fig. 3. Evolução da Medida LCOM nas 10 Versões do Sistema de Software FindBugs

Fatores, por exemplo, maturidade, podem diminuir com o aumento do valor da medida LCOM, uma vez que um sistema maduro deve ter valores elevados de coesão. Como pode ser observado na Fig. 3, a medida LCOM apresentou pouca variação, porém constante crescimento até a versão 1.3.8, com valores entre 137,56 e 143,84. Em seguida, o valor dessa

[3] http://findbugs.sourceforge.net/
[4] http://findbugs.sourceforge.net/downloads.html
[5] https://netbeans.org/kb/docs/java/code-inspect.html
[6] https://code.google.com/p/findbugs-idea/
[7] http://wiki.hudson-ci.org/display/HUDSON/FindBugs+Plugin
[8] https://wiki.jenkins-ci.org/display/JENKINS/FindBugs+Plugin
[9] https://eclipse.org

medida teve substancial aumento entre as versões 1.3.9 e 2.0.0 com variação entre 156,94 e 165,21. Nas versões seguintes, o valor da medida LCOM voltou a apresentar "leve" crescimento. O valor da medida LCOM para a versão 3.0.0 foi 167,26. Analisando esses valores, verificou-se que há baixa coesão durante a evolução do FindBugs para as 10 versões analisadas. A última versão apresenta valor superior em relação à primeira versão analisada.

Como pode ser observado na Fig. 4, a medida TCC apresentou aumento até terceira versão 1.3.7 na qual foi verificado valor aproximando-se de 0,22. Em seguida, o sistema de software apresentou queda entre as cinco próximas versões, com exceção da última versão que, apesar de ter aumento, obteve valores menores que a sua primeira versão, o qual foi de aproximadamente de 0,21. A queda de valores de coesão em um sistema significa que algo "não vai bem", pois eles podem influenciar na qualidade. Altos valores de coesão estão diretamente relacionados com maturidade, confiabilidade e estabilidade essenciais para a qualidade de um software.

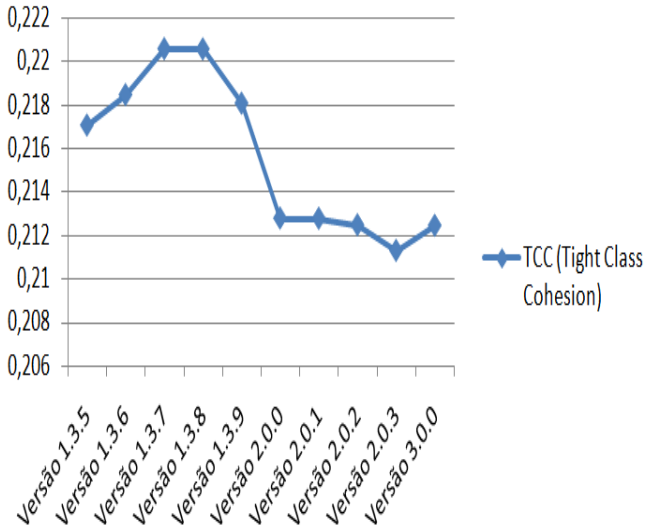


Fig. 4. Evolução da Medida TCC nas 10 Versões do Sistema de Software FindBugs

Em função da queda de coesão, as medidas de acoplamento CBO, DAC e MPC apresentaram aumento em maior parte da evolução do sistema de software, porém com suas particularidades, como pode ser observado na Fig. 5, na Fig. 6 e na Fig. 7, respectivamente.

As medidas CBO e DAC, apesar de terem valores diferentes, apresentaram queda somente na versão 2.0.2, os valores aproximaram-se de 6,59 e 5,85, respectivamente. Os valores mínimos de ambas as medidas foram encontrados na primeira versão e os valores máximos na última versão, sendo que o menor e o maior valores registrados para a medida CBO foram 6,31 e 6,73, respectivamente, e o menor e o maior valores registrados para a medida DAC foram 5,64 e 5,98, respectivamente. A medida MCP apresentou crescimento, com valores entre 9,62 em sua primeira versão e 11,15 em sua última versão. Apesar desse crescimento, a medida MPC apresentou ligeira queda na versão 2.0.2, comportamento observado no valor das medidas DAC e CBO, e na última versão.

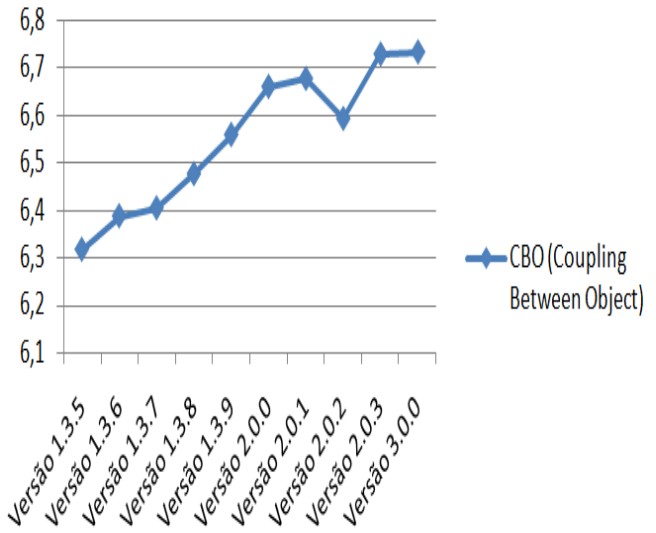


Fig. 5. Evolução da Medida CBO nas 10 Versões do Sistema de Software FindBugs

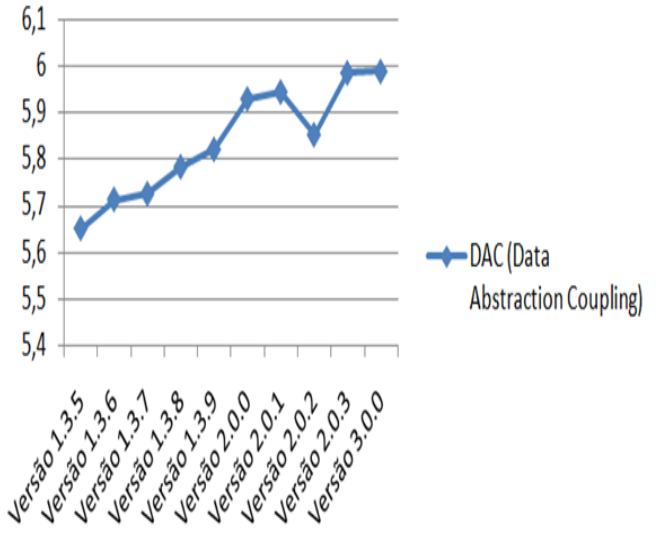


Fig. 6. Evolução da Medida DAC nas 10 Versões do Sistema de Software FindBugs

As medidas Ce e Ca apresentaram comportamento diferente das demais medidas de acoplamento, como pode ser observado na Fig. 8 e na Fig. 9, respectivamente. Essas medidas apresentaram crescimento contínuo até a quinta versão (versão 1.3.9) e, em seguida, apresentaram queda na versão 2.0.0, em que os valores registrados foram Ce = 11,28 e Ca = 24,94. Após a versão 2.0.0, as duas medidas apresentaram aumento na versão 2.0.1 e queda na versão 2.0.2 e, em seguida, aumento nas duas últimas versões.

*1) Correlação entre as Medidas*

As medidas analisadas no estudo podem auxiliar no monitoramento e na melhoria da evolução de sistemas de

software desde o início do desenvolvimento. Características como alta coesão e baixo acoplamento são indicadores de qualidade; deste modo, uma correlação entre as medidas de acoplamento e de coesão é essencial [18]. Como intuito de verificar se elas influenciaram na qualidade desses sistemas de alguma forma, a correlação entre as medidas de acoplamento e de coesão analisadas é apresentada na Table III. Em seguida, o resultado é discutido.

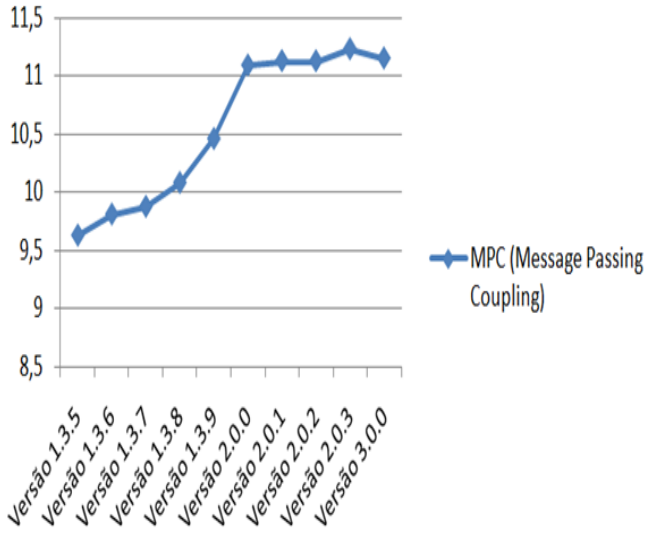


Fig. 7. Evolução da Medida MPC nas 10 Versões do Sistema de Software FindBugs

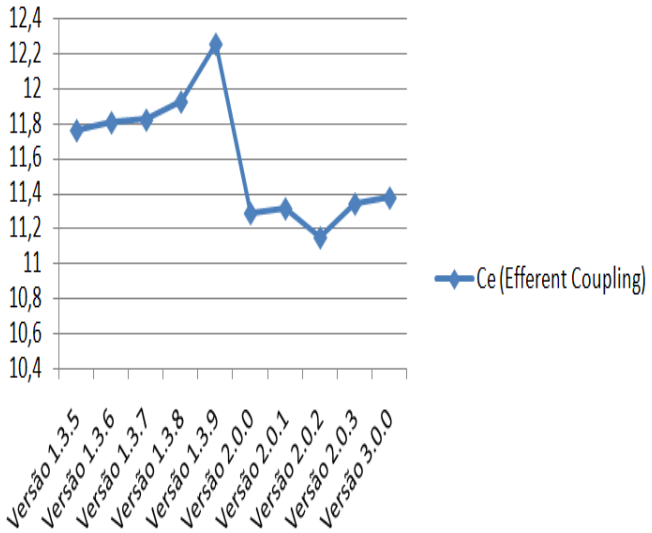


Fig. 8. Evolução da Medida Ce nas 10 Versões do Sistema de Software FindBugs

Qualidade é um conceito subjetivo que pode ser obtida analisando vários aspectos, características e particularidades de um produto, processo ou fator que se deseja avaliar. Considerando as 10 versões analisadas do FindBugs, os resultados apresentam indícios que a versão de melhor qualidade foi a primeira, considerando as medidas LCOM, CBO e DAC mostradas na Fig. 3, na Fig. 5 e na Fig. 6, respectivamente. Essas medidas foram as que apresentaram menor oscilação entre os dados coletadas.

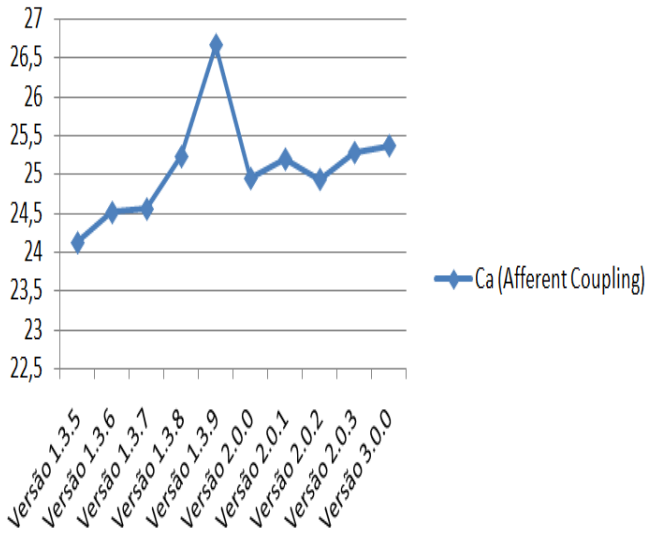


Fig. 9. Evolução da Medida Ca nas 10 Versões do Sistema de Software FindBugs

TABLE III. VALORES DO COEFICIENTE DE CORRELAÇÃO ENTRE OS VALORES DE COESÃO E DE ACOPLAMENTO

| | | LCOM | TCC |
|---|---|---|---|
| **CBO** | **CC** | 0,959 | -0,816 |
| | **p** | **0,000** | **0,004** |
| **DAC** | **CC** | 0,938 | -0,812 |
| | **p** | **0,000** | **0,004** |
| **MPC** | **CC** | 0,994 | -0,890 |
| | **p** | **0,000** | **0,001** |
| **Ce** | **CC** | -0,699 | 0,857 |
| | **p** | **0,024** | **0,002** |
| **Ca** | **CC** | 0,469 | -0,101 |
| | **p** | 0,172 | 0,781 |

CC: Coeficiente de Correlação; p: nível de significância.
Valores de p em negrito indicam valor de correlação significativo.

As medidas CBO e DAC apresentaram crescimento contínuo e praticamente idêntico considerando apenas uma queda em ambas na versão 2.0.2, que se aproximou de valores da versão 1.3.9. A medida LCOM também teve o mesmo comportamento, apresentando queda na versão 3.0.0. O coeficiente de correlação apresentado na Table III apresenta valores significativos para as medidas CBO e LCOM (0,959) e para as medidas DAC e LCOM (0,938) com nível de significância $p < 0,01$ com queda na qualidade. Isso aconteceu, pois, uma vez que as medidas DAC e CBO são influenciadas negativamente pelo acoplamento, parte do sistema pode ser utilizada por outras partes, o que dificulta o entendimento do código e diminui a compreensão à medida que o valor das medidas DAC e CBO aumentam.

Em contrapartida, a medida TCC apresentou aumento até a terceira versão, ocorrendo melhoria do sistema de software, dado que a qualidade está ligada a valores altos de coesão.

Fatores, por exemplo, maturidade, confiabilidade e compreensibilidade, podem ter valores melhores como o aumento do valor da medida TCC. Em seguida, essa medida apresentou queda entre as versões 1.2.8 e 2.0.3 e aumentou na última versão, embora apresentando valores mais baixos do que as primeiras versões. Alta coesão e baixo acoplamento são desejados; assim, forte correlação entre as medidas DAC e TCC é essencial [18].

Como pode ser observado na Table III, correlação foi registrada entre essas medidas (-0,812) com nível de significância $p < 0,01$. Esse fato apresenta indícios que o sistema de software deve ser alterado, pois essa correlação apresenta valor elevado para a medida DAC e baixo valor para a medida TCC. O comportamento dessas medidas pode ser observado na Fig. 10.

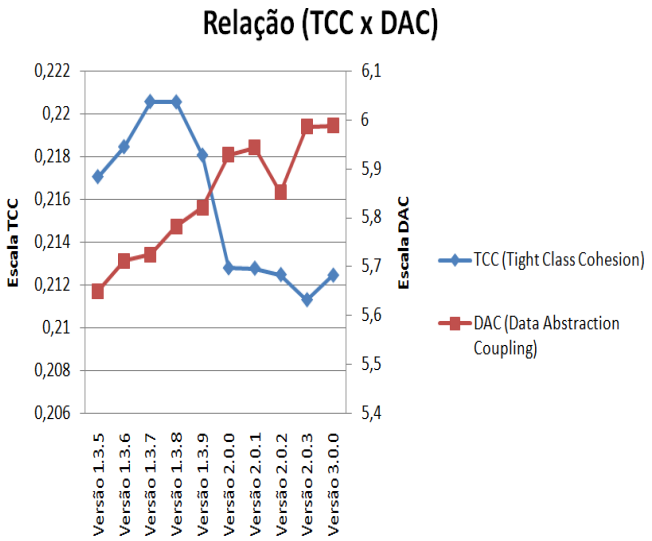


Fig. 10. Relação Medidas TCC x DAC nas 10 Versões do Sistema de Software FindBugs

A medida MPC apresentou crescimento contínuo ao longo da evolução do sistema de software com características particulares. Foi registrada queda simultânea no valor das medidas MCP e DAC na versão 2.0.2 e no valor das medidas MCP e CBO na versão 3.0.0, como pode ser observado na Fig. 11. Foi verificado que as medidas CBO, DAC e MCP apresentaram correlação linear positiva ($p < 0,01$) com a medida LCOM e correlação linear negativa ($p < 0,01$) com a medida TCC. Assim, quanto mais acoplado o sistema de software se apresentou, menor foi seu valor de coesão. Apesar da medida LCOM apresentar correlação positiva, ela mede a falta de coesão; diante desse fato, quanto maior o valor dessa medida, menos coeso é o sistema de software.

Apesar da falta de coesão registrada no decorrer das versões do FindBugs, ao correlacionar a medida Ce e as medidas TCC e LCOM, elas apresentaram comportamento inesperado a partir da versão 2.0.0. O comportamento da medida TCC em relação à medida Ce pode ser observado na Fig. 12. Assim, essas medidas assumem valor de coeficiente de correlação negativo e positivo, respectivamente. Ao verificar essas medidas, quanto menor o valor de acoplamento, maior foi o valor de coesão. Os valores da medida Ce diminuíram, sendo que o menor valor apresentado foi na versão 2.0.2. Apesar dessa medida aumentar nas versões seguintes, esses valores ficaram abaixo da primeira versão. Esse fato pode-se referir à mudança da estrutura ou da programação no desenvolvimento do sistema de software.

A medida Ca foi a única que não apresentou correlação com as demais. Apesar de graficamente as medidas Ce e Ca apresentarem semelhança no comportamento, elas não apresentaram correlação linear significativa ($p > 0,05$).

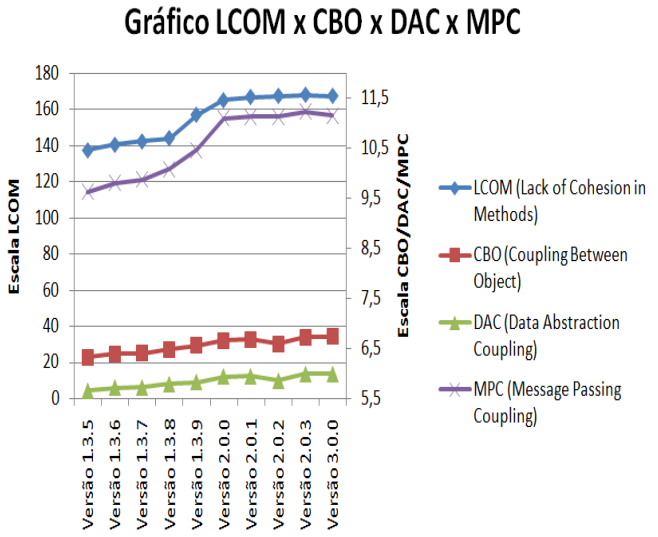


Fig. 11. Relação entre as Medidas LCOM, CBO, DAC e MPC nas 10 Versões do Sistema de Software FindBugs

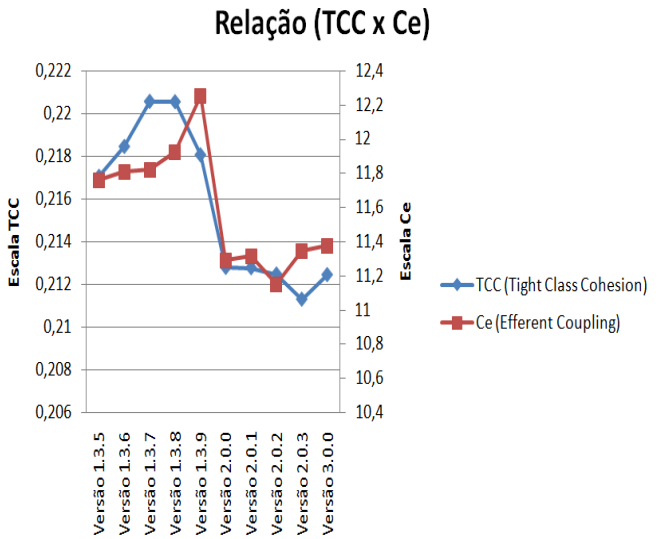


Fig. 12. Relação Medidas TCC x Ce obtidos segundo 10 versões do `FindBugs`

## V. AMEAÇAS A VALIDADE

Um ponto importante a ser ressaltado é estudo analisar apenas as 10 ultimas versões funcionais, consecutivas e

disponíveis de um sistema de software (FindBugs) desenvolvido em uma linguagem de programação específica (Java). Deste modo, os resultados obtidos podem não caracterizar toda a vida útil do sistema de software FindBugs, bem como não ser generalizável para qualquer sistema de software desenvolvido com a linguagem de programação Java ou em outra linguagem.

## VI. Trabalhos Relacionados

Neste trabalho, como em outros encontrados na literatura, foram realizados análise e acompanhamento da evolução de sistemas de software via medidas, entretanto cada um com suas particularidades e contribuições.

Uma análise integrada das características de sistemas de software como um todo e de cada modificação individual feita durante seu ciclo de vida foi realizada em um trabalho [20]. Foram coletadas medidas básicas e derivadas as quais foram representadas por gráficos. Esses gráficos apresentaram a variação dos valores das medidas em relação ao tempo. Com esse estudo, houve a possibilidade de extração do valor de medidas de forma transparente aos usuários que desejam testar e estudar seus próprios sistemas de software desenvolvidos em Java.

Em outro estudo [16], foi apresentada uma abordagem para a observação das medidas de código fonte, estudando-as utilizando suas distribuições e suas associações, além de discutir as relações de causalidade e implicações práticas-gerenciais para seu monitoramento. Foram estudadas a distribuição e as correlações dos valores das medidas de 38 sistemas de software de código aberto. Uma das contribuições dessa pesquisa foi uma análise detalhada em relação ao comportamento, aos valores e aos estudos de caso de medidas de código fonte e à diminuição das contradições das análises realizadas.

Em seguida, considerando o trabalho anterior, em outro trabalho [4], foram realizadas análises de medidas de forma integrada, permitindo que essas análises fossem realizadas de forma mais ampla. Nesse trabalho, foi apresentada uma proposta de integração de medidas estáticas e dinâmicas, visto que a medição é uma forma de avaliar a qualidade de sistemas de software. Os resultados obtidos permitiram uma visão mais completa desses sistemas ainda na fase de modelagem, tornando mais claro o seu entendimento e proporcionando, no início do ciclo de vida, *feedback* para a melhoria da qualidade do software.

Em um estudo mais recente [10], foi realizada uma revisão sistemática para expor a eficiência das medidas presentes na suíte CK. Algumas medidas, tais como, CBO, RFC e WMC, foram utilizadas com sucesso em todos os estudos realizados, enquanto as demais medidas (LCOM, DIT e NOC) obtiveram sucesso em apenas parte dos estudos. Além disso, foi elaborada uma tabela com valores dessas medidas a fim de apontar possíveis problemas do sistema de software. Por fim, esse estudo propôs um novo modelo de visualização de sistemas de software baseado em uma metáfora simplista, que auxilia na compreensão, transformando as classes de um sistema em corpos celestes que possuem características estipuladas de acordo com o valor de cada métrica.

Nos estudos anteriores várias medidas foram utilizadas para verificar a qualidade e apontar possíveis problemas em sistemas de software. Em outro trabalho [8], foram utilizadas medidas de acoplamento, as quais auxiliam na identificação dos elementos que possam impactar na qualidade de sistemas de software orientados a objetos. Os autores apresentaram resultados de um estudo com desenvolvedores de software de diferentes níveis de experiência e avaliaram o grau de acoplamento de um projeto. Com base nessas respostas, foram relacionadas medidas de acoplamento com o propósito de identificar pontos críticos e melhorar a característica de qualidade manutenibilidade de sistemas de software orientados a objetos.

## VII. Considerações Finais

Com os dados coletados e analisados, os gráficos gerados apresentam a evolução do sistema de software FindBugs, identificando aspectos não detectados no decorrer do processo de desenvolvimento. Foi verificado que algumas medidas de acoplamento possuem correlação com as medidas de coesão. Assim, as medidas CBO, DAC, MPC influenciam diretamente nas medidas LCOM e TCC. Desse modo, essas medidas influenciaram diretamente o sistema de software FindBugs, apresentando características indesejáveis como alto acoplamento e baixa coesão na maior parte das medidas analisadas comprometendo a qualidade.

Com os resultados apresentados neste trabalho, é possível oferecer *feedback* para os desenvolvedores, possibilitando verificar cada aspecto do FindBugs e oferecer melhorias para as suas futuras versões. Isso pode ressaltar mais uma vez a importância da medição de sistemas de software não somente em sua versão final, mas em todo o processo de desenvolvimento.

Com sugestões de trabalhos futuros, por exemplo, outras ferramentas estatísticas para verificar a correlação entre as medidas analisadas podem ser utilizadas, outras medidas podem ser utilizadas na análise da evolução de sistemas de software e outros sistemas de software devem ser analisados. Além disso, analisar sistemas de software desenvolvidos em outras linguagens de programação, bem como outras propriedades, além da coesão e do acoplamento.

## Referências

[1] Abreu, B. F.; Melo, W. (1996) Evaluating the Impact of Object Oriented Design on Software Quality. In: International Symposium on Software Metrics, pp 90-99.

[2] Bansiya, J.; Davis, C. G. (2002) A Hierarchical Model for Object Oriented Design Quality Assessment. In: IEEE Transactions on Software Engineering, pp. 4-17.

[3] Bär, H.; Bauer, M.; Ciupke, O.; Demeyer, S.; Ducasse, S.; Lanza, M.; Marinescu, R.; Nebbe, R.; Nierstrasz, O.; Richner, T.; Rieger, M.; Riva, C.; Sassen, A. M.; Schulz, B.; Steyaert, P.; Tichelaar, S.; Weisbrod, J. (1999) The FAMOOS Object-Oriented Reengineering Handbook. Technical Report, Forschungszentrum Informatik, Karlsruhe, Software

Composition Group, University of Berne, ESPRIT Program Project 21975.

[4] Barreto, D. L.; Barcelos, M. R. S.; Vasconcelos, A. P. V. (2012) Integração de Métricas Estáticas e Dinâmicas para Apoiar a Avaliação da Qualidade em Modelos de Software. In: IX Workshop de Manutenção de Software Moderna.

[5] Bieman, J. M.; Kang, B.-K. (1995). Cohesion and Reuse in an Object-Oriented System. In: SIGSOFT Software Engineering Notes. 20 pp. 259–262.

[6] Briand, L.; Daly, W.; Wust, J. (1999) A Unified Framework for Coupling Measurement in Object-Oriented Systems. In: IEEE Transactions on Software Engineering, pp. 91-121.

[7] Chidamber, S. R.; Kamerer, C. F. (1994) A Metrics Suite for Object-Oriented Design. In: IEEE Transactions on Software Engineering, pp. 476-493.

[8] Rodrigues, B. R. de O.; Souza, D. E. de; Figueiredo, E. M. L. (2014) Medindo Acoplamento em Software Orientado a Objeto: Uma Perspectiva do Desenvolvedor. In: Abakós. v. 3, n. 1. pp. 3-17.

[9] Eick, S. G.; Graves, T. L.; Karr, A. F.; Marron, J. S.; Mockus, A. (2001) Does Code Decay? Assessing the Evidence from Change Management Data. In: IEEE Transactions on Software Engineering, pp. 1-12.

[10] Juliano, R. C. (2014) Visualização de Software Baseada em uma Metáfora do Universo Utilizando o Conjunto de Métricas CK. Dissertação de Mestrado - Universidade Federal de Uberlândia. 124p.

[11] Lanza, M.; Marinescu, R. (2006) Object-Oriented Metrics in Practice. Springer.

[12] Lehman, M. M. (1969) The Programming Process. Technical report, IBM Research Division.

[13] Lehman, M. M. (1996) Laws of Software Evolution Revisited". In: European Workshop on Software Process Technology, pp. 108-124.

[14] Li, W.; Henry, S. Object-Oriented Metrics that Predict Maintainability. In: Journal of Systems and Software. V. 23, I. 2, pp. 111-122. 1993.

[15] Martin, R. C. (1994) Design Quality Metrics - An Analysis of Dependencies. In: Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA.

[16] Meirelles, P. R. M. (2013) Monitoramento de Métricas de Código Fonte em Projetos de Software Livre. Tese Doutorado (Ciência de Computação). Universidade de São Paulo, p. 161.

[17] Mens, T.; Demeyer, S. (2008) Software Evolution. Springer.

[18] Panas, T. R.; Lincke, J.; Lundberg, W.; Lowe, A. (2005) Qualitative Evaluation of a Software Development and Re-Engineering Project. In: Annual IEEE/NASA Software Engineering Workshop.

[19] R Development Core Team. R A Language and Environment for Statistical Computing. Vienna: Fundation for Statistical Computing, 2009.

[20] Ribeiro, W. S.; Ribeiro, D. D. C.; Plastino, A.; Murta, L. G. P. (2012) Acompanhamento da Evolução de Software via Métricas. In: Workshop de Manutenção de Software Moderna (WMSWM).

[21] Scacchi, W. (2003) Understanding Open Source Software Evolution. In: Applying, Breaking, and Rethinking the Laws of Software Evolution. John Wiley and Sons Inc.