Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Nelson Barbosa Junior

Avaliação da Manutenibilidade de *Software*

Através das Métricas de C&K

São Paulo
2011

Nelson Barbosa Junior

Avaliação da Manutenibilidade de *Software*

*A*través das Métricas de C&K

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, como parte dos requisitos para a obtenção do título de Mestre em Engenharia da Computação.

Data da aprovação: 30 de Agosto de 2011.

_____

Prof. Dr. Kechi Hirama (Orientador)
Escola Politécnica da Universidade de São Paulo

Membros da Banca Examinadora:

Prof. Dr. Kechi Hirama (Orientador)
Escola Politécnica da Universidade de São Paulo

Profa. Dra. Selma Shin Shimizu Melnikoff (Membro)
Escola Politécnica da Universidade de São Paulo

Prof. Dr. João Batista de Camargo Jr. (Membro)
Escola Politécnica da Universidade de São Paulo

Nelson Barbosa Junior

Avaliação da Manutenibilidade de *Software*

*A*través das

Métricas de C&K

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo como parte dos requisitos para obtenção do título de Mestre em Engenharia de Computação.

Área de Concentração: Engenharia de *Software*.

Orientador: Prof. Dr. Kechi Hirama

São Paulo
Agosto/2011

Ficha Catalográfica
Elaborada pelo Departamento de Acervo e Informação Tecnológica – DAIT
do Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT

B238a **Barbosa Junior, Nelson**

Avaliação da manutenibilidade de software. Nelson Barbosa Junior. São Paulo, 2011.
108 p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Engenharia de Software

Orientador: Prof. Dr. Kechi Hirama

1. Manutenção de software 2. Métrica de software 3. Análise estatística 4. Tese I. Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Coordenadoria de Ensino Tecnológico II. Título

11-66 CDU 004.416(043)

DEDICATÓRIA

Aos meus pais Nelson (in memoriam) e Izabel (in memoriam).

Ao meu filho Leonardo.

À minha esposa Maria Alice.

AGRADECIMENTOS

À minha amada esposa Maria Alice que, durante toda a duração deste trabalho foi companheira, amiga, me ouviu e apoiou nos momentos bons e, principalmente, nos difíceis.

Ao meu orientador Professor Kechi Hirama, pela competência, pelo apoio e pela paciência.

RESUMO

Avaliar a manutenibilidade de um *software* de forma rápida e antecipada pode trazer ganhos em diversos níveis do projeto à medida que se podem executar, o quanto antes, ações preventivas ou corretivas em seu planejamento, por exemplo, fazendo com que recursos valiosos sejam preservados ou aplicados em áreas mais específicas ou em fases posteriores. O objetivo do trabalho proposto, baseado em um experimento, é de avaliar a manutenibilidade de determinado *software*, através da análise estatística dos valores das métricas propostas por Chidamber & Kemerer (C&K), obtidos a partir do código-fonte escrito em Java. O método utilizado no trabalho prevê a montagem de ambientes computacionais contendo versões distintas do código-fonte do *software*, para suportar os eventos de coleta dos valores das métricas que servirão como fonte para a análise estatística dos dados obtidos. A contribuição maior é, neste caso, investigar o relacionamento entre as métricas de C&K e a manutenibilidade, identificando possíveis tendências da manutenibilidade do *software* através da análise estatística, com base nos valores das métricas obtidos com as versões diferentes e subsequentes de seu código-fonte.

Palavras-chave: métricas de *software*; manutenibilidade de *software*; medidas.

ABSTRACT

Evaluation of Software Maintainability using the C&K Metrics

Assessing the software maintainability quickly and early can generate earnings in many levels of the project as they can run, as soon as possible, preventive or corrective actions in their planning, for example, so that valuable resources are preserved or applied in more specific areas or in later stages. The objective of this work, based on an experiment is to evaluate the maintainability of specific software, through statistical analysis of values of the metrics proposed by Chidamber & Kemerer (C&K), and obtained from the Java source code. The method used in this work need mounting of computing environments containing different versions of the software source code, to support the collection the values of the metrics that will serve as a source for statistical analysis of data obtained. The greater contribution is in this case, investigate the relationship between C&K metrics and maintainability, identifying possible trends maintainability of software through statistical analysis, based on the metrics values obtained through different and subsequent source code versions.

Keywords: software metrics; software maintainability; measures.

Lista de Ilustrações

Lista de Tabelas

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo o trabalho é contextualizado, com a apresentação do tema foco do trabalho e a motivação para desenvolvê-lo, complementado com o que tem sido feito em relação ao tema, ressaltando o que é preciso verificar, bem como as contribuições esperadas.

1.1 Motivação

Desde quando se iniciou a história dos sistemas de *software* em 1945 (ZVEGINTZOV e PARIKH, 2005, p. 1), iniciou-se também a história da manutenção de sistemas de *software* que se chamará, daqui em diante, simplesmente *software*.

Manutenibilidade – medida do quão fácil um sistema de *software* ou componente pode ser modificado para corrigir falhas, melhorar seu desempenho ou seus atributos, ou adaptar-se às alterações do ambiente (IEEE STD 610, 1990, p. 46), confiabilidade, usabilidade, eficiência, funcionalidade e portabilidade são atributos de qualidade externos do *software* (ISO/IEC 9126-1, 2001, p. 13). À medida que o *software* recebe novos requisitos, ou é corrigido, torna-se mais complexo podendo desviar-se do seu projeto original, levando-o, por exemplo, a uma manutenibilidade pobre (REDDY e RAO, 2009, p. 1) exigindo maiores esforços durante seus testes.

Sabe-se que uma das ações para controlar algo é medi-lo, já que não se consegue controlar aquilo que não se consegue medir (DEMARCO, 1986). No caso do *software* desenvolvido com base nos conceitos de orientação a objetos, foco deste trabalho, uma das maneiras de se controlar seu ciclo de vida é pelo uso de ferramentas capazes de obter os valores de suas métricas, que permitem monitorar, controlar e melhorar a forma como se desenvolve e se faz sua manutenção (BASILI, BRIAND e MELO, 1996, p. 1).

As métricas podem ser estáticas quando, pelo uso de ferramentas que processam o código-fonte, calcula-se uma série de valores estáticos sem que os *softwares* estejam em execução (SINGH e SINGH, 2008, p. 1). Também podem ser dinâmicas, quando, pelo uso de ferramentas que analisam o comportamento dos *software*s durante sua execução, calcula-se uma série de valores característicos (SINGH e SINGH, 2008, p. 2).

Alguns conjuntos de métricas procuram caracterizar as propriedades internas do *software* como, por exemplo, abstração, polimorfismo, encapsulamento, herança, acoplamento e coesão, diretamente a partir do código-fonte do *software*, representando uma maneira de exteriorizar, na forma de seus atributos de qualidade, como está a organização de suas estruturas internas. Porém, um dos problemas do uso de métricas é que não se consegue avaliar muitos dos atributos externos do *software* enquanto não se estiver bastante avançado no processo de desenvolvimento do código ou do produto como um todo (EMAM, 2001, p. 3).

Um dos conjuntos de métricas estáticas mais citados em pesquisas foi proposto por Chidamber & Kemerer, pela primeira vez, em 1991 (CHIDAMBER e KEMERER, 1991). Por simplicidade, é utilizada, a partir daqui, a mesma forma de citação encontrada na literatura para Chidamber & Kemerer que é C&K e, quando citadas, subentende-se que são métricas estáticas. O trabalho foi introduzido pelos autores como sendo um conjunto inicial de seis candidatas a métricas especialmente desenvolvidas para a medição do tamanho e da complexidade de *softwares* desenvolvidos com base em OO - Orientação a Objetos (CHIDAMBER e KEMERER, 1991). Cada métrica apresentada tem alguma relação com a definição do objeto, com os atributos ou com a comunicação com o objeto.

Em 1994 os mesmos autores divulgaram um novo trabalho sobre métricas (CHIDAMBER e KEMERER, 1994). Nele, indicam a existência de críticas a respeito da ausência de teorias sólidas que apoiam as métricas, sua complexidade no uso e sua aplicação. Propuseram, então, a base teórica e a correspondente validação prática necessárias à justificativa do uso das métricas propostas.

Porém, não se identificou até o momento nenhuma técnica, ou ferramenta, ou mecanismo, que possibilite visualizar ligação clara entre os valores das métricas propostas por C&K, indicadores das propriedades internas do *software*, com seus atributos de qualidade externos (CHAHAL e SINGH, 2009, p. 1), em particular junto à manutenibilidade do *software*.

Sendo o conjunto de métricas proposto por C&K voltado para *softwares* desenvolvidos com base no conceito de orientação a objetos (CHIDAMBER e KEMERER, 1991, p. 9) é necessário encontrar pacotes contendo códigos-fonte desenvolvidos em linguagens de programação orientada a objeto ou, pelo menos, que possuam suporte a orientação a objetos. C++, C#, Java, Object Pascal, Python, Ruby e Smaltalk são exemplos de linguagens de programação orientadas a objeto (GROGONO, 2002), enquanto PHP, Perl, VB.NET são exemplos de linguagens de programação com suporte a orientação a objetos. Neste trabalho é adotada a linguagem Java em função da variedade de pacotes de códigos-fonte disponíveis em repositórios na internet, com o acesso livre, tal como o *SourceForge* (http://sourceforge.net).

Lembrando que o *software* é um produto lógico que não pode ser tocado, ou visto (LANZA, GALL e DUGERDIL, 2009, p. 294), torna-se relevante o uso de alguma ferramenta, ou mecanismo, que auxilie na avaliação de suas propriedades externas com base nos valores de suas métricas. Qualquer que seja a ferramenta, o mecanismo, a técnica de representação gráfica, ou mesmo algum tipo de análise estatística precisa favorecer, por exemplo, a compreensão da manutenibilidade, entre outros atributos externos do *software*, com o menor esforço.

1.2 Objetivo

Este trabalho objetiva avaliar a evolução da manutenibilidade de um *software* através de versões subsequentes de um código-fonte, tendo por base a análise estatística dos valores obtidos pelas métricas propostas por C&K.

1.3 Justificativas

A elaboração deste trabalho traz uma forma diferente de avaliar a manutenibilidade de um *software*. O que se encontra na literatura e é mais comum, é a aplicação conceitual de uma ou mais métricas com o objetivo de se avaliar alguma característica daquele *software* e naquele ponto de seu desenvolvimento. A mudança que se fará neste trabalho é reunir diversas avaliações das propriedades internas do mesmo *software* através dos valores das métricas de C&K obtidos por um processo semiautomático, verificar como

a manutenibilidade daquele *software* vem se manifestando e avaliar alguma projeção futura da manutenibilidade daquele *software*.

Nas pesquisas realizadas há trabalhos relacionados como o de Chidamber *et al* (CHIDAMBER, DARCY e KEMERER, 1998) que desenvolve uma avaliação das métricas com base em um modelo de código com sete classes e propõe funções exploratórias relacionadas à produtividade, por exemplo, porém não se aproxima da manutenibilidade.

Há também o trabalho de Lincke *et al* (LINCKE, LUNDBERG e LÖWE, 2008) que propõe algumas hipóteses e as confronta com a ajuda de várias ferramentas para cálculo das métricas de C&K, comparando os resultados dos valores das métricas obtidos com cada ferramenta, além da proposição do cálculo da manutenibilidade com base em pesos, mas de uma maneira não muito clara.

O trabalho de Tang *et al.* (TANG, KAO e CHEN, 1999) também propõe uma análise estatística dos valores das métricas de C&K, porém voltada à detecção de defeitos.

Kanellopoulos *et al.* (KANELLOPOULOS, DIMOPULOS, *et al.*, 2006) obtém os valores das métricas, faz a análise estatística, mas limita-se a apenas uma versão de um código-fonte.

O trabalho de Selvarani *et al.* (SELVARANI, NAIR e PRASAD, 2009) apresenta pesquisas empíricas registradas em tabelas e referentes à identificação da propensão a defeitos de alguns projetos, com base em apenas três das seis métricas de C&K.

Já no trabalho de Yu *et al.* (YU e ZHOU, 2010) há cálculos que procuram relacionar coesão e acoplamento com defeitos. Mas exclui a maioria das métricas de C&K.

1.4 Resultados Esperados e Contribuições

O conjunto de valores das métricas de um *software* pode ser uma ferramenta para identificar e analisar aspectos obscuros de *softwares* complexos (LANZA e MARINESCU, 2006, p. 4).

Com este trabalho, espera-se avaliar a manutenibilidade de um determinado *software* com base nos valores das métricas que são obtidos com o uso de ferramenta específica para seu cálculo os quais, em seguida, passam por uma análise estatística que relaciona os valores das métricas com os principais conceitos que envolvem a codificação OO – Orientação a Objetos.

Uma contribuição deste trabalho é apresentar um conjunto de tabelas com os valores das métricas calculadas por ferramentas com acesso irrestrito, o que não se encontra com facilidade na literatura.

1.5 Método de Trabalho

A elaboração deste trabalho abrangeu as seguintes atividades:

- Pesquisa bibliográfica em busca de embasamento teórico necessário tanto ao pacote de métricas proposto por C&K, como à manutenibilidade de *software* e à eventual correlação entre estes temas;
- Pesquisa de ferramentas que forneçam os valores das métricas propostas por C&K;
- Pesquisa de formas de correlacionar ou identificar a dependência entre os valores das métricas;
- Pesquisa bibliográfica sobre como classificar um código-fonte como bom, e quais as boas práticas do mercado para essa classificação quando o código-fonte é escrito em Java;
- Pesquisa de *softwares* com documentos e código-fonte abertos para se criar os ambientes necessários ao experimento e coletar os valores das métricas correspondentes;
- Calcular e tabular os valores das métricas, interpretando-os.

1.6 Organização do Trabalho

O trabalho está organizado da seguinte forma:

- O capítulo um, Introdução, descreve o problema em seu contexto, relacionado com a dificuldade de avaliar a manutenibilidade de *software* utilizando métricas; apresenta as métricas propostas por Chidamber &

Kemerer; e sugere a forma baseada em gráficos para avaliar a correlação entre manutenibilidade e os valores das métricas;

- O capítulo dois, Métricas de *Software*, apresenta a compilação das informações dos trabalhos mais representativos dentre os pesquisados e referentes às métricas de *softwares* desenvolvidos com base nos conceitos de orientação a objetos, como obtê-las e quantificá-las, privilegiando o conjunto de métricas proposto por Chidamber & Kemerer;
- O capítulo três, Manutenibilidade de *Software*, discorre de forma breve sobre este atributo de qualidade de *software*, conceituando-o e relacionando-o com o modelo da norma ISO/IEC 9126, além de apresentar um conjunto de características que permitem classificar como bom ou ruim determinado código escrito em Java.
- O capítulo quatro, Ferramentas para Coleta de Medidas, apresenta as características necessárias às ferramentas que se propõem fazer a coleta das medidas das métricas e as justificativas para a ferramenta escolhida;
- O capítulo cinco, O Processo de Medição e Análise, apresenta onde e como o código-fonte foi encontrado e quais os critérios utilizados, mostra como foram obtidos os valores das métricas através da ferramenta escolhida e explora possível correlação entre as métricas obtidas através da Correlação de Pearson.
- O capítulo seis, Experimento e Análise dos Dados, apresenta os dados obtidos e a respectiva análise.
- O capítulo sete, Conclusão, apresenta os pontos de vista a que se chegou diante da observação dos dados obtidos e sugere possíveis trabalhos futuros.

2 MÉTRICAS DE *SOFTWARE*

Neste capítulo são conceituadas e apresentadas algumas métricas citadas na literatura, com aplicação junto ao desenvolvimento de *software* com base em orientação a objeto. Dentre os trabalhos pesquisados, os mais relevantes para este capítulo foram os de Ordonez *et al*. (2008) que conceitua medições, medidas e métricas, juntamente com as normas IEEE 610 e IEEE 1061, Genero *et al.* (2005) que recomendam cuidados no uso das métricas, Harrison *et al.* (1997) que pesquisaram os trabalhos mais significativos sobre métricas, Chidamber & Kemerer (1994) que detalham seu pacote de métricas, de Subramanyam *et al*. (2003) com sua análise empírica das métricas C&K.

2.1 Introdução

A prática em torno das métricas envolve suas definições, seus valores e indicadores (ORDONEZ e HADDAD, 2008, p. 1).

Métrica é uma função cujas entradas são os dados do *software* e cuja saída é um único valor numérico que pode ser interpretado como o grau em que o *software* possui um dado atributo de qualidade (IEEE STD 610, 1990, p. 60).

O indicador representa, por exemplo, uma informação útil sobre processos e suas possíveis melhorias resultantes da aplicação de métricas (ORDONEZ e HADDAD, 2008, p. 1).

Métricas são, tipicamente, divididas em três categorias: de processos, de projetos e de produtos (DASKALANTONAKIS, 1992, p. 999), (ORDONEZ e HADDAD, 2008, p. 2), (HONGLEI, WEI e YANAN, 2009, p. 131). Aquelas relativas a processos são usadas para avaliar a produtividade individual, ou de um time, ou de uma organização, com o objetivo final de melhorar os processos relacionados ao desenvolvimento e manutenção de *software*. O processo para se atingir as melhorias envolve a medição de atributos especificamente relacionados com o processo de desenvolvimento e manutenção de *software*, a posterior criação de um conjunto de métricas baseadas nos atributos previamente definidos e, por fim, a utilização das medidas obtidas como fornecedoras de indicadores que direcionem a estratégia em direção à melhoria desejada. Um exemplo prático da aplicação

das métricas de processos é a busca de uma melhor eficiência, individual ou de uma equipe, na identificação, no mapeamento e na eliminação de defeitos de *software*, fornecendo melhores respostas para a correção e testes (ORDONEZ e HADDAD, 2008, p. 3).

As métricas que se voltam a projetos procuram avaliar o planejamento, as características e a execução dos projetos, gerando informações úteis para gerentes e desenvolvedores ajustarem suas atividades, além de, muitas vezes, colaborarem com o desenvolvimento das métricas de processos. As medidas das métricas de projetos anteriores são usadas para subsidiar as estimativas de tempo e esforço dos novos projetos que, com o monitoramento de cada projeto em curso, devem se ajustar ao longo do tempo. São incluídas nesta categoria, por exemplo, a quantidade de linhas de código (LOC – *Lines of Code*), pontos de função, métricas de acoplamento e coesão (ORDONEZ e HADDAD, 2008, p. 3).

Por último, as métricas de produto, foco deste trabalho, procuram avaliar as características ou atributos do produto de *software*, ou parte dele, ou de um componente específico, incluindo aquelas relacionadas ao código-fonte e aos testes, por exemplo. As aplicações práticas destas métricas avançam sobre as áreas da especificação do produto, na arquitetura, na complexidade, no desempenho e manutenibilidade, entre outras (ORDONEZ e HADDAD, 2008, p. 3).

As primeiras métricas surgiram em 1974 pela necessidade de medir e comparar a taxa de produção de linhas de código (LOC, em inglês) dos programadores (ZUSE, 1998, p. 3) e deveriam contribuir com o planejamento, com o monitoramento, com o controle e com a avaliação da qualidade tanto dos produtos como dos processos de *software* (BRIAND, MORASCA e BASILI, 1994, p. 1).

Medidas podem se obtidas diretamente, tais como a quantidade de linhas de código, ou a quantidade de defeitos por pontos de função, ou de forma indireta, tais como a confiabilidade, a complexidade e a manutenibilidade (PRESSMAN, 2001, p. 509).

Há muitas propostas sobre o assunto métricas de *software*, tornando o assunto suficientemente extenso para ser abordado de maneira completa em apenas um trabalho. Dentre as propostas estudadas e referentes a projetos desenvolvidos com base em orientação a objetos, três delas são citadas com frequência em (HARRISON, COUNSELL e NITHI, 1997, p. 230) e correspondem aos seguintes trabalhos:

- *A Metric Suite for Object Oriented Design* (CHIDAMBER e KEMERER, 1994) ou C&K;
- *Object-Oriented Software Metrics* (LORENZ e KIDD, 1994) ou L&K;
- *Toward the Design Quality Evaluation of Object-Oriented Software Systems* (ABREU, GOULÃO e ESTEVES, 1995) ou Abreu.

Este trabalho focará nas métricas de C&K já que, com base em outros trabalhos (BASILI, BRIAND e MELO, 1996), (EMAM, BENLARBI, *et al.*, 2001), (KJELLQVIST, 2008), (EMAM, 2001), (SUBRAMANYAM e KRISHNAN, 2003), (KULKARNI, KALSHETTY e ARDE, 2010) permite-se afirmar que as métricas de C&K possuem base teórica suficiente que justificam seu uso, assim como a existência de protótipos e produtos comerciais que permitem coletá-las.

2.2 Métricas de C&K

Um dos conjuntos de métricas mais citados em pesquisas foi proposto por C&K, pela primeira vez, em 1991 (CHIDAMBER e KEMERER, 1991). Esse artigo foi introduzido pelos autores como sendo um conjunto inicial de seis candidatas a métricas especialmente desenvolvidas para a medição do tamanho e da complexidade de *softwares* desenvolvidos com base em OO - Orientação a Objetos. Cada métrica apresentada tem alguma relação com a definição do objeto, com os atributos do objeto ou com a comunicação com o objeto.

Ainda no trabalho de C&K (CHIDAMBER e KEMERER, 1991), as métricas apresentadas e seu suporte teórico visavam fornecer ajuda às seguintes áreas do gerenciamento:

- Estimar custos e calendário para futuros projetos;
- Avaliar o impacto na produtividade de novas ferramentas e técnicas;

- Estabelecer as tendências da produtividade ao longo do tempo;
- Melhorar a qualidade do *software*;
- Prever futuras necessidades da equipe;
- Antecipar e reduzir futuros requisitos de manutenção.

Em 1994 os mesmos autores divulgaram um novo trabalho sobre métricas (CHIDAMBER e KEMERER, 1994). Nele, indicam a existência de críticas a respeito da ausência de teorias sólidas que apoiam as métricas, sua complexidade no uso e sua aplicação.

Ao longo dos trabalhos pesquisados encontram-se citações referentes a algumas questões que devem ser consideradas na definição das métricas de *software*. Genero *et al.* (GENERO, PIATTINI e CALERO, 2005) resumiram em cinco itens estas questões. O primeiro item descrito refere-se à definição da referida métrica, que deve ser clara. O segundo item refere-se aos objetivos da métrica que devem ser claramente definidos. O terceiro item é a ausência ou presença da validação teórica da métrica. O quarto item a considerar é a existência, ou não, da validação empírica da métrica. E, por último, é o item referente à existência de ferramentas específicas para as métricas propostas, evidenciando se existe, ou não, a possibilidade de cálculo automático dos valores das métricas.

Considerando também as críticas recebidas, C&K propuseram, em 1994, uma base teórica e a correspondente validação prática necessárias à justificativa do uso das métricas propostas.

O conjunto de métricas propostas por C&K, em 1994 é:

1. Métodos Ponderados por Classe (ou WMC, *Weighted Methods per Class*): seu valor é obtido pela contagem dos métodos implementados na classe ou através da soma das complexidades dos métodos, sendo que esta complexidade é medida através da Complexidade Ciclomática desenvolvida por Thomas J. McCabe (MCCABE, 1976). Este valor coincide com a simples contagem do

total dos métodos[1], caso se considere cada método igualmente complexo e de valor igual a um[2], sendo que, em geral, a medida desta métrica deve ser menor que 10. Está ligada diretamente à definição dos objetos e aos atributos dos objetos.

2. Profundidade da Árvore de Herança (ou *DIT, Depth of Inheritance Tree*): é o valor correspondente à maior distância entre a classe raiz e o nó mais distante. Quanto mais profunda na hierarquia a classe estiver, maior o número de métodos que possivelmente tenha herdado, tornando mais complexo prever seu comportamento. Está ligada diretamente à definição dos objetos.

3. Número de Classes Filhas (ou *NOC, Number of Children*): seu valor equivale ao número de descendentes imediatos de uma classe. É um indicador da influência em potencial da classe sobre o sistema como um todo. Está ligada diretamente à definição dos objetos.

4. Acoplamento entre Classes de Objetos (ou *CBO, Coupling Between Objects*): seu valor é calculado pela contagem do número de classes com as quais uma classe está acoplada, tanto como cliente, como fornecedor de informação. Quanto maior for o número de acoplamentos maior é a sensibilidade a mudanças em outras partes do projeto, tornando a manutenibilidade mais difícil. Não se pode esquecer que o acoplamento entre classes pode ser alto, dentro de um pacote (*package*), enquanto o acoplamento entre pacotes pode ser baixo. Está ligada diretamente à comunicação entre os objetos.

5. Respostas para uma Classe (ou *RFC, Response for a Class*): envolve a combinação da complexidade da classe através da quantidade de métodos e a comunicação com outros objetos. Seu valor é calculado pela contagem do número de métodos que podem ser executados em resposta a uma mensagem recebida. Quanto

[1] Nas fases iniciais do projeto, embora as classes e métodos estejam definidos, ainda não estão implementados. Por isso, o valor da métrica WMC não pode ser obtido nesta fase. Uma das soluções para esta medição é contar o número de métodos existentes na classe em estudo, o que a transforma em uma métrica de tamanho e não mais de complexidade (KULKARNI, KALSHETTY e ARDE, 2010, p. 646).

[2] Pode-se usar o valor zero para o WMC no caso de funções privadas e independentemente de sua complexidade (TANG, KAO e CHEN, 1999).

maior for o número de métodos invocados por um objeto através de mensagens, maior a complexidade de sua classe. Em geral quanto maior o acoplamento, maiores os custos das atividades de manutenção. Está ligada diretamente aos atributos dos objetos e à comunicação entre os objetos.

6. Falta de Coesão (ou *LCOM, Lack of Cohesion in Methods*): seu valor pode ser calculado como o número de pares de métodos sem variáveis compartilhadas menos o número de pares com variáveis compartilhadas, considerando-se como zero caso o valor seja negativo. A falta de coesão aumenta a complexidade. Está ligada diretamente aos atributos dos objetos e é uma espécie de métrica reversa da coesão. Em 1996, Henderson-Sellers (HENDERSON-SELLERS, 1996, p. 142) revisou a definição original, simplificando-a e normalizando-a.

O relacionamento entre as medidas das características dos objetos e as métricas de C&K (KULKARNI, KALSHETTY e ARDE, 2010, p. 2) são as que estão apresentadas na Tabela 1:

Tabela 1: Relacionamento entre medidas de projeto OO e as métricas C&K.

Medidas de Projeto OO	**Métricas C&K**
Classe	WMC, RFC, LCOM
Atributo	LCOM
Método	WMC, RFC, LCOM
Herança	DIT, NOC
Coesão	LCOM
Acoplamento	CBO, RFC

2.3 Métricas de C&K e Atributos de Qualidade

Muitos dos trabalhos sobre as métricas de C&K buscaram ligações diretas e indiretas com os atributos de qualidade do *software*.

O trabalho de Subramanyam *et al.* (SUBRAMANYAM e KRISHNAN, 2003, p. 299) traz um resumo dos resultados das buscas destas ligações, apresentado na Tabela 2, com base em nove trabalhos diferentes (coluna "Estudo") que estudaram parcial ou integralmente as métricas propostas por C&K (coluna "Métricas Testadas"), gerando um conjunto de ligações entre as métricas e alguns atributos de qualidade (coluna "Resultados"), tais como a manutenibilidade e a testabilidade.

Tabela 2: Resumo dos resultados dos trabalhos sobre as métricas de C&K e suas ligações com atributos de qualidade de *software*.

Estudo	Métricas Testadas	Resultados
Wei,L; Sallie,H 1993	Todas	Com exceção do CBO, as outras cinco métricas ajudaram a prever o esforço sobre a manutenção.
Basili *et al.* 1996	Todas	Somente LCOM não se relacionou com defeitos.
Subramanyam (2003 apud [Binkley & Schach 1998])	CBO, NOC	As medidas de acoplamento se relacionaram com a necessidade de manutenção, mas a NOC não.
Chidamber, S.R.; Darcy, D.P.; Kemerer, C.F. - 1998	Todas	Alto valor de CBO e baixo valor de LCOM foram associados com maior retrabalho, baixa produtividade e maior esforço no projeto.
Briand, L.C.; Morasca, S.; Basili, V.R. - 1999	CBO, RFC, LCOM	As três métricas foram relacionadas com a predisposição das classes a falhas.
Tang, M.H.; Kao, M. H.; Chen, M.H.-1999	WMC, RFC	Altos valores de WMC e RFC foram relacionados com a predisposição a falhas.
Subramanyam (2003 apud [Briand *et al.* 2000])	Todas	Altos valores de WMC, CBO, DIT e RFC foram relacionados com predisposição a falhas, enquanto que as classes com maior NOC foram menos propensas a falhas. LCOM não foi associado a defeitos.
Subramanyam (2003 apud [Cartwright & Shepperd 2000])	DIT, NOC	Identificou-se a influência das duas métricas na densidade de defeitos por linha de código.
El-Emam, K. 2001	Todas	Se uma associação é encontrada entre uma métrica em particular e a tendência a falhas, pode ser devido ao fato de que os valores mais elevados desta métrica também são devidos a um maior tamanho do aplicativo, trazendo confusão à métrica.

Com base nos conceitos de cada métrica proposta e os artigos pesquisados, identificam-se as seguintes recomendações entre este pacote de métricas e alguns dos principais atributos de qualidade (CHIDAMBER, DARCY

e KEMERER, 1998), (BORGES, 2006), (SATC-SOFTWARE ASSURANCE TECHNOLOGY CENTER, 1995) como indicado na Tabela 3.

Tabela 3: Métricas de C&K, sua relação com os alguns atributos de qualidade e valores recomendados.

Medida	Atributos de Qualidade	Valores Recomendados e Características
WMC	Complexidade, Manutenibilidade e Compreensibilidade	<= 20
DIT	Reusabilidade, Testabilidade	<= 6 Alto: reuso restrito ou não explorado Alto: aumenta complexidade
NOC	Reusabilidade, Testabilidade	Alto: reuso não explorado Alto: poucas características comuns Alto: problemas estruturais
CBO	Manutenibilidade, Testabilidade, Reusabilidade	Alto: restringe reuso Alto: maior esforço nos testes Alto: baixa dependência
RFC	Complexidade, Testabilidade	Alto: maior complexidade Alto: maior capacidade de resposta
LCOM		A ausência de coesão é algo indesejável Não associada a defeitos

Essas ligações foram obtidas (CHIDAMBER, DARCY e KEMERER, 1998), (BORGES, 2006), de acordo com os trabalhos, de forma empírica, coletando-se os valores de cada métrica, ora de forma manual, ora de forma automatizada. No entanto, um dos pontos ainda incompleto nesses trabalhos é a indicação da evolução de cada métrica ao longo do processo de desenvolvimento, ou de manutenção, já que o os mesmos trabalhos dão a entender que os valores obtidos referem-se ao retrato final do *software*.

Entende-se que, ao avaliar a evolução de cada métrica, pode ser possível identificar que um valor já foi muito maior, ou muito menor, ao longo do desenvolvimento do *software* e pode ter influenciado outras métricas e, indiretamente, outros atributos de qualidade.

Além disso, um valor de uma métrica isoladamente pode não significar que o *software* como um todo está sendo construído de forma incorreta. É preciso avaliar todos os valores obtidos e de forma conjunta.

À medida que se avalia a evolução contínua do conjunto de métricas, pode-se identificar a existência de gatilhos, isto é, disparadores de uma reação, sejam para valores aceitáveis ou inaceitáveis de determinada característica do *software*. Esses gatilhos podem ajudar a identificar, de forma antecipada, projetos com estruturas carentes ou programas escritos de forma pobre (EMAM, 2001, p. 7). Mas se a análise não for feita de forma contínua, progressiva, à medida que novas versões são desenvolvidas ou compiladas, o que se obtém ao final é apenas um valor para a métrica, ou conjunto delas. Nesta situação, caso não tenha havido o acompanhamento contínuo não se saberá se o atributo de qualidade relacionado a aquela métrica evoluiu ou involuiu, já que a informação disponível – a métrica, retrata a situação no momento de sua medição, sem indicar as tendências dos resultados das influências de cada métrica entre si e do todo sobre cada uma, não permitindo avaliar adequadamente a situação de alguns atributos de qualidade, em particular, a manutenibilidade.

2.4 Considerações do Capítulo

Os trabalhos até aqui pesquisados indicam que os valores obtidos pelas diversas métricas apenas sugerirão quais são os atributos de qualidade de *software* mais recomendados a se avaliar (HARRISON, COUNSELL e NITHI, 1997, p. 234). A complexidade, a reusabilidade, a manutenibilidade e a testabilidade são citados com frequência.

Com ajuda da análise estatística dos valores das métricas, pretende-se identificar tendências e padrões que permitam avaliar a manutenibilidade à medida que considera algumas das principais características do *software* desenvolvido com base em orientação a objeto.

3 MANUTENIBILIDADE DE *SOFTWARE*

Neste capítulo são explorados os conceitos e as características de manutenção e manutenibilidade como atributo de qualidade de *software*, fatores que podem influencia-los e o relacionamento com as medidas das métricas de C&K, com base nos trabalhos pesquisados. Os principais trabalhos envolvidos são de Zhuo *et al*. (1993) que explora modelos de avaliação de manutenibilidade e o resultado de suas aplicações, de Coleman *et al*. (1994) que explora outras partes dos modelos, de Aggarwal *et al*. (2002) que explora os fatores influenciadores da manutenibilidade de *software*, e de Antonellis *et al*. (2007) que traz um trabalho específico entre as métricas de C&K e seu relacionamento com a manutenibilidade. Também é apresentado um conjunto de tópicos que buscam contribuir com um bom código escrito em Java.

3.1 Introdução

As diversas atividades do processo de manutenção de *software*, ou suas partes, tem sido uma das mais críticas e longas partes do ciclo de vida dos sistemas de *software* (HUNG e ZOU, 2005, p. 1). Caper Jones define 21 atividades diferentes abaixo do termo genérico manutenção (JONES, 2006, p. 5). Durante as atividades deste processo as equipes despendem boa parte de seu tempo na investigação, no entendimento do código-fonte, na forma como está organizado e na interação entre suas partes, principalmente quando a documentação do *software* foi perdida, ou está desatualizada, ou está inacessível. Este cenário ainda permanecerá desafiante porque prever a condição da manutenibilidade do *software*, de forma antecipada, por exemplo, já na fase de projeto, ajudará no processo de alteração da arquitetura do *software* em busca de um melhor desempenho e na redução dos custos da própria manutenção (MUTHANNA, KONTOGIANNIS, *et al.*, 2000, p. 8). Além disso, a manutenibilidade, que só pode ser medida de forma indireta (ANDA, 2007, p. 2), sofre influência do projeto e dos princípios de arquitetura adotados que passam por mudanças ao longo do tempo à medida que a tecnologia também muda. A manutenibilidade, então, pode ser encarada como uma forma indireta de comparar alternativas de projetos (ALAGAR, LI e ORMANDJIEVA, 2001, p. 2).

3.2 Manutenção e Manutenibilidade

Neste trabalho, utilizam-se, como definições, aquelas dadas pela IEEE 610, tanto para Manutenção de *Software*, como para Manutenibilidade de *Software*. Assim, “manutenção de *software* é definida como o processo de modificação do sistema de *software* ou seus componentes, depois de suas entregas, para a correção de defeitos, melhoria de desempenho ou outros atributos, ou adaptação ao ambiente no qual está inserido” (IEEE STD 610, 1990, p. 46). Por simplicidade é usado apenas manutenção a partir deste ponto. Na mesma norma encontra-se “manutenibilidade de *software* como sendo o quão fácil um sistema de *software* ou seus componentes pode ser modificado para corrigir defeitos, melhorar seu desempenho ou outros atributos, ou adapta-lo ao ambiente no qual está inserido” (IEEE STD 610, 1990, p. 46). Também, por simplicidade, é usada apenas manutenibilidade neste trabalho.

Na literatura encontram-se, comumente, quatro divisões para a manutenção. Aliás, cinco divisões quando é incluída a recomendada por Thomas Pigoski (PIGOSKI, 1997, p. 81). Estas divisões seriam (VERHOEF, 2000, p. 2):

- Manutenção anterior à entrega (*Predelivery Maintenance*, em inglês), voltada para as atividades que acontecem antes da entrega do *software*;
- Manutenção Corretiva, voltada para a correção de defeitos latentes ou falhas do *software*;
- Manutenção Adaptativa, voltada para a adaptação do *software* exigida pela mudança no ambiente no qual está inserido ou por uma evolução funcional;
- Manutenção Perfectiva, voltada para a melhoria do desempenho;
- Manutenção Preventiva, voltada para a prevenção de problemas antes que ocorram.

O primeiro tipo - *Predelivery Maintenance* - foi proposto por Thomas Pigoski (PIGOSKI, 1997, p. 81) dentro do contexto da norma ISO/IEC 12207 que indica a necessidade de determinadas atividades de manutenção antes

que o *software* seja entregue e que são do interesse da manutenibilidade, tais como o planejamento da logística para suportar o *software*, a garantia da suportabilidade e o planejamento de sua transição após a entrega. Os três tipos seguintes – Manutenção Corretiva, Adaptativa e Perfectiva - foram propostos por E. Burton Swanson (SWANSON, 1976). E o último tipo da lista – Manutenção Preventiva - foi proposto na norma *Standard for Software Maintenance* (IEEE STD 1219, 1993).

3.3 Modelos de Manutenibilidade de *Software*

Em relação à manutenibilidade, dada sua definição, utilizam-se, neste trabalho, as subdivisões propostas pela norma ISO/IEC 9126 e seu Relatório Técnico sobre métricas (ISO/IEC 9126-2, 2002), com o objetivo final de se alcançar as medidas da manutenibilidade. Na literatura encontram-se referências a vários modelos para a medida da manutenibilidade. Dentre eles há, por exemplo, aquele que se propõe a medir a manutenibilidade com base no número de alterações no código ao longo de determinado período (VAN KOTEN e GRAY, 2005, p. 4). Outros modelos, classificados como de previsão do esforço de manutenção, se propõem a medir o esforço da manutenção requerida pelo *software*, com base em medidas obtidas, de forma integrada, diretamente em artefatos do *software* como o código-fonte e a documentação (AGGARWAL, SINGH e CHHABRA, 2002, p. 235). Sendo o código-fonte o principal artefato de onde se fazem as medições que alimentarão a medida da manutenibilidade, Aggarwal *et al*. (2002) indicam três fatores que afetam a manutenibilidade: a qualidade da documentação, a facilidade na leitura e a compreensibilidade do código-fonte sendo que, este último, conforme o *SWEBOK* (2004) representa entre 40% e 60% dos esforços do processo de manutenção (ABRAN, MOORE, *et al.*, 2004, p. 6-4). A manutenibilidade, então, é a tradução dos descobrimentos técnicos feitos ao nível dos artefatos relacionados com o código-fonte em avaliações representativas do *software* como um todo (KANELLOPOULOS e HEITLAGER, 2008, p. 1).

Coleman *et al*. (1994), citado em vários dos trabalhos pesquisados, explora os modelos de avaliação hierárquica multidimensional, de regressão polinomial, da medida da complexidade agregada, da análise dos componentes principais

e do fator de análise (COLEMAN, LOWTHER, *et al.*, 1994). Apesar da variedade de modelos em geral, a exatidão na previsão da manutenibilidade é baixa (ELISH e ELISH, 2009, p. 69), independentemente do modelo.

Zhuo *et al*. (1993) acrescenta a esta lista o modelo de estimativa da manutenibilidade pela complexidade, e classifica os modelos, entre si, da seguinte maneira: o modelo de avaliação hierárquica multidimensional é mais fácil de ser usado, o modelo de análise dos componentes principais é mais preciso e mais complicado de ser usado que o modelo regressão polinomial, enquanto que este último pode ser utilizado na automatização de técnicas estatísticas (ZHUO, LOWTHER, *et al.*, 1993).

Com base no modelo proposto pela ISO/IEC 9126, existem métricas externas, que são aquelas observadas durante a execução do *software*, existem medidas internas, que são aquelas obtidas sem a execução do *software*, e aquelas medidas classificadas como qualidade em uso, relacionadas com as propriedades do *software* durante sua operação e manutenção. A manutenibilidade é dada por uma das seis métricas externas, ou atributos externos de qualidade propostos no modelo de qualidade de *software* da ISO/IEC 9126, que também se subdivide em cinco partes:

- Analisabilidade: o quão fácil ou difícil é diagnosticar o *software* para identificar deficiências ou causas de falhas, ou para identificar partes a serem modificadas?
- Modificabilidade: o quão fácil ou difícil é fazer alterações no *software*?
- Estabilidade: o quanto é fácil ou difícil manter o *software* em um estado estável durante seu processo de alteração?
- Testabilidade: o quão fácil ou difícil é testar o *software* após ter sofrido alterações?
- Conformidade: o quanto o *software* está ou não está de acordo com normas e convenções relativas à manutenibilidade?

A relação entre o modelo ISO/IEC 9126 e um conjunto genérico de métricas é apresentada na Figura 1 (ANTONELLIS, ANTONIOU, *et al.*, 2007, p. 3).

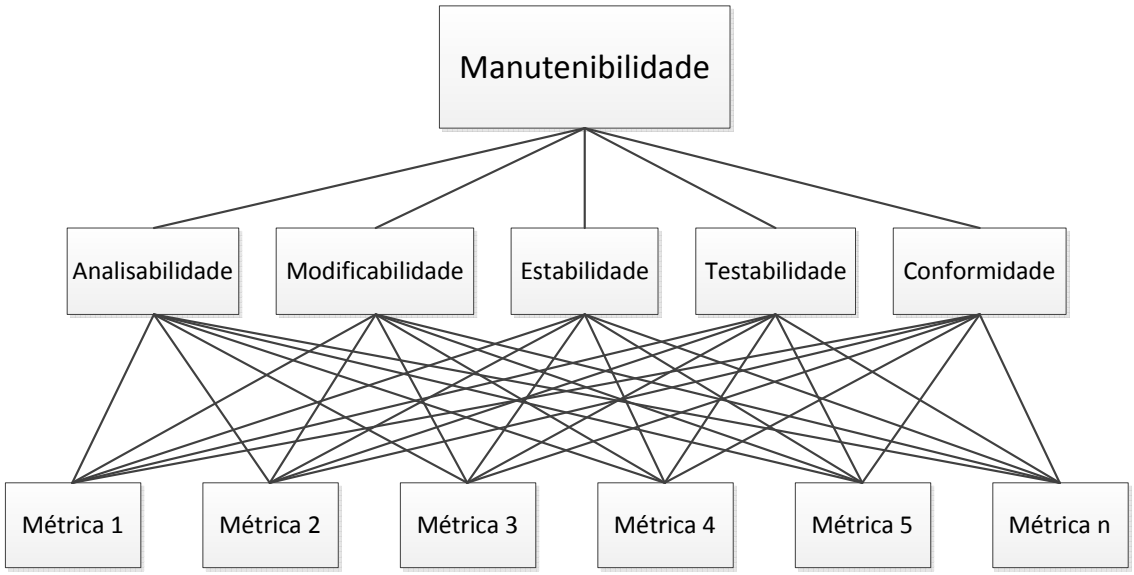

Figura 1: Modelo genérico relacionando a manutenibilidade como atributo de qualidade, ou métrica externa, e as métricas internas de *software*, retirado da norma ISO/IEC 9126-2.

Ao tentar utilizar as métricas de C&K conforme o modelo da Figura 1 observa-se em Kulkarni *et al*. (2010) que nem todas as métricas propostas por C&K se relacionam com os atributos e características do modelo, conforme a Tabela 4.

Tabela 4: Relacionamento entre atributos de qualidade e as métricas de C&K **(KULKARNI, KALSHETTY e ARDE, 2010, p. 647)**.

Fatores de Qualidade	**Métricas C&K**
Compreensibilidade	RFC, CBO, DIT
Reusabilidade	WMC, CBO, DIT, NOC
Testabilidade	RFC, CBO, NOC
Manutenibilidade	WMC, CBO
Esforço de Desenvolvimento	WMC, LCOM

Ainda em Kulkarni *et al*. (2010) para compensar a falta de relacionamento entre as métricas C&K originais, vários autores modificam ou complementam o

conjunto original de métricas para manter o relacionamento com os fatores de qualidade, considerando eventuais redundâncias entre as métricas. Esta complementação, juntamente com a aplicação de pesos às métricas escolhidas, de acordo com processo de avaliação hierárquica multidimensional (ZHUO, LOWTHER, *et al.*, 1993, p. 62) permitiu a aplicação das métricas de C&K na avaliação da manutenibilidade dos *softwares* utilizados nos ensaios (ANTONELLIS, ANTONIOU, *et al.*, 2007, p. 5).

3.4 Um Bom Código Escrito em Java

Para que produza efeito à aplicação do conjunto de métricas escolhido, seja ele qual for, sobre o código-fonte de um *software* em estudo, é necessário que este código-fonte seja avaliado de forma antecipada (ANDERSSON e VESTERGREN, 2004, p. 29) com o foco voltado para a qualidade do código em si. Este foco, aliás, deve acompanhar todo o processo, particularmente durante a fase de escolha do código-fonte para estudo.

Nos trabalhos pesquisados não foram encontradas indicações de maneiras definitivas para que um bom código em Java seja produzido. Encontram-se, na realidade, recomendações sobre as melhores práticas para a construção de códigos, até porque há questões de arquitetura, segurança e encapsulamento, por exemplo, específicas para cada *software*, que devem ser respeitadas, inviabilizando uma maneira única de se construir o código do *software*. Resta, porém, a opção de comparação entre códigos, ou suas partes, e para isto não há necessidade de sofisticação para classificar uma peça de código como melhor que outro. Na Tabela 5, retirada do sítio http://www.virtualmachinery.com/jhawkmetrics.htm, estão descritos duas peças de código que produzem o mesmo resultado.

Tabela 5: Apresentação de duas peças de código escrito em Java, através dos quais se pode comparar qual deles é considerado o melhor código (http://www.virtualmachinery.com/jhawkmetrics.htm, acessado em 09 de Junho de 2011 às 10h).

public static double zzz(double x, double z) {
z+=(z*x/100);
return z;
}
	(1)

public static double addInterestToBalance(double interestRate, double balance) {
/* Add interest to balance */
balance+=(balance*interestRate/100);
return balance;
}
	(2)

O sítio[3] afirma que o código apresentado na parte inferior da Tabela 5, parte (2), é melhor que o código apresentado na parte superior, parte (1), simplesmente porque é mais fácil de ser lido e indica ao leitor o que está sendo executado. Por exemplo, na parte (1) a expressão “z+=(z*x/100)” produz o mesmo resultado que a expressão “balance+=(balance*interestRate/100)” que está na parte (2) e indica de forma mais visível as operações e as variáveis envolvidas.

Nos trabalhos pesquisados, as recomendações sobre as melhores práticas não se resumem apenas à elaboração do código em si, mas também de técnicas e métodos relacionados aos processos que apoiam a elaboração do código-fonte. Por exemplo, ao se utilizar práticas ágeis de desenvolvimento, uma boa prática é elaborar o código usando duplas de programadores (*pair programming*, em inglês) (STAPEL, LÜBKE e KNAUSS, 2008, p. 1). O uso de duplas de programadores permite que ambos trabalhem juntos e colaborando no desenvolvimento das mesmas tarefas, com o objetivo de melhorar a qualidade final do *software* (MADEYSKI, 2006).

[3] Acesso realizado em 09 de Junho de 2011 às 10h.

3.5 Um Código Ruim Escrito em Java

Uma das maneiras para se medir a manutenibilidade é através do Índice de Manutenibilidade (*MI, Maintainability Index*, em inglês), não adaptado à orientação a objetos. Este índice baseia-se no *HV-Halstead Volume* (HALSTEAD, 1977) previamente composto pelo número de operadores e operandos no código, na CC-Complexidade Ciclomática (MCCABE, 1976), na média da quantidade de linhas de código por módulo (LOC-*Lines of Code*) e, opcionalmente, pela porcentagem de linhas de comentário por módulo (HEITLAGER, KUIPERS e VISSER, 2007, p. 32). Com os mesmos números de operadores e operandos, pode-se obter outra medida, denominada “Dificuldade de Halstead”. Para altos valores desta medida, quando adaptada à orientação a objeto, há indicações da existência de um código ruim (ANDERSSON e VESTERGREN, 2004, p. 52). Nestas situações, tanto as medidas, como as métricas geradas devem ser rejeitadas.

Tanto em *Code Complete* (MCCONNELL, 2004), como em *Refactoring: Improving the Design os Existing Code* (FOWLER, BECK, *et al.*, 1999) há citações de justificativas frequentes para a inclusão de comentários no código do *software* para poder explicar o código devido à sua má forma de organização e dificuldade de entendimento. Ou seja, o comentário compensa a possível forma ruim que o código possui.

Um código mal escrito, além de ser uma potencial fonte de problemas latentes, reflete uma má distribuição do trabalho dos desenvolvedores ao longo do tempo. É frequente a situação em que o desenvolvedor dedica duas horas para depurar um código que demandou 30 minutos para ser escrito (MCCONNELL, 2004, p. 592). Provavelmente teria sido melhor reescreve-lo.

3.6 Considerações do Capítulo

O conhecimento sobre a manutenibilidade do *software* é algo viável e verificável. Por este atributo de qualidade um extrato das características essenciais do *software* podem ser medidas por um conjunto de métricas, focadas principalmente no código-fonte, independentemente do tipo de manutenção que o *software* pode estar recebendo, combinando-se em apenas um indicador representando a manutenibilidade. Independentemente do tipo de

manutenção que estiver sendo praticada, algumas das aplicações deste índice híbrido são (i) a avaliação dos custos da manutenção de *software*, (ii) controle da qualidade e dos esforços do desenvolvimento e (iii) como um mecanismo para fazer cumprir as normas de manutenção antes da aceitação ou entrega (ZHUO, LOWTHER, *et al.*, 1993, p. 61).

Em particular a métrica WMC, integrante do conjunto de métricas de C&K, permite explorar tanto a manutenibilidade, como a reusabilidade da classe, sendo que, quanto maiores são seus valores, mais difícil é sua manutenibilidade e reusabilidade (ANTONELLIS, ANTONIOU, *et al.*, 2007, p. 4).

4 FERRAMENTAS PARA A COLETA DOS VALORES DAS MÉTRICAS

Neste capítulo, descrevem-se algumas das ferramentas pesquisadas para a coleta dos valores das métricas, bem como parte do processo a ser aplicado. O principal trabalho referenciado capítulo é o de Lincke *et al*. (2008) que apresenta uma análise de 36 ferramentas distintas, elabora critérios de seleção e resume em 10 as ferramentas adequadas para a obtenção das métricas de C&K.

4.1 Introdução

Entende-se que uma ferramenta para a coleta dos valores das métricas seja um programa de computador que tem implementado em si as definições dos conjuntos de métricas que se pretende utilizar na avaliação de um atributo de *software*, por exemplo, a manutenibilidade. Isto não quer dizer, no entanto, que todas as ferramentas fornecerão todos os valores referentes a todas as métricas esperadas, e nem que todo o processo seja automático. Há, nos trabalhos pesquisados, de diversos fornecedores, vários tipos de ferramentas e com perfis de aplicação distintos.

Neste trabalho, as pesquisas foram direcionadas para a busca de ferramentas que oferecessem resultados diretos baseados nos conceitos das métricas de C&K.

4.2 Escolha das Ferramentas

Com base no trabalho de Lincke *et al*. (2008), foram consideradas ferramentas que apresentassem os seguintes perfis:

- Trabalham com a linguagem Java;
- Forneçam informações sobre as métricas de C&K, mesmo que parcialmente;
- Permitem seu uso sem custos financeiros diretos;
- Aquelas que, mesmo sem custos financeiros diretos, não se limitam a uma quantidade pequena de linhas de código.

Com base nestes critérios, foram encontradas 10 ferramentas conforme a Tabela 6 (LINCKE, LUNDBERG e LÖWE, 2008, p. 3).

Tabela 6: Lista com as 10 ferramentas para coleta dos valores das métricas de C&K, que atendem aos critérios descritos em Lincke *et al*. (2008), juntamente com os endereços eletrônicos dos sítios de seus desenvolvedores.

Ferramentas	Métricas								
	CBO	DIT	LCOM-C&K	LCOM-HS	NOC	NOM	RFC	LOC	WMC
Analyst4j http://www.codeswat.com/cswat/index.php?option=com_content&task=view&id=43&Itemid=63	X	X	X		X	X	X		X
CCCC http://sourceforge.net/projects/cccc	X	X			X	X			
CKJM - Chidamber & Kemerer Java Metrics http://www.spinellis.gr/sw/ckjm/	X	X	X		X		X		X
Dependency Finder http://depfind.sourceforge.net/		X			X	X			
Eclipse Metrics Plugin 1.3.6 http://sourceforge.net/projects/metrics		X		X	X	X			X
Eclipse Metrics 3.4 http://eclipse-metrics.sourceforge.net/			X	X					X
OOMeter http://www.ccse.kfupm.edu.sa/~oometer/oometer	X	X	X		X			X	
Semmle http://semmle.com/download/		X	X	X	X	X	X		
Understand for Java http://www.scitools.com/products/understand/java/metrics.php	X	X	X		X	X			
VizzAnalyser http://w3.msi.vxu.se/~tps/VizzAnalyzer/	X	X	X		X	X	X	X	X

De acordo com a Tabela 6, é possível obter valores das métricas de C&K (CBO, DIT, LCOM, NOC, RFC e WMC), de Henderson-Sellers (LCOM-HS) e a quantidade de linhas de código (LOC), sendo que as duas últimas não são relevantes para este trabalho porque não se incluem nas seis métricas de C&K.

Das ferramentas citadas na Tabela 6 apenas a ferramenta OOMeter não se encontrava disponível à época da elaboração deste trabalho[4].

Depois de avaliar os pré-requisitos de cada ferramenta decidiu-se pelo uso, neste trabalho, da ferramenta “CKJM - Chidamber & Kemerer Java Metrics” pelos seguintes motivos:

- Fornece todas as seis métricas de C&K, entre outras, simultaneamente para cada classe;
- Trabalha diretamente com o *bytecode* (arquivos *.class*) dos arquivos Java, compilados, de cada classe, o que dispensa a necessidade do código-fonte;
- O processamento do código é rápido, gerando resultados de imediato sem exigir grande capacidade instalada de *hardware*;
- Independe da existência de um ambiente de desenvolvimento, como o Eclipse, por exemplo, estruturado de forma isenta de erros durante o processo de carregamento do código para posterior compilação;
- Outros pré-requisitos restringem-se ao sistema operacional que pode ser Windows ou Linux, que tenham uma versão da Java VM 1.5 ou superior;
- Pode ser acionada através de linhas de comando facilmente montadas incluindo argumentos para a exportação direta dos resultados;
- Os resultados obtidos podem ser capturados diretamente da tela ou importados, com a ajuda de arquivos-texto, para planilhas eletrônicas, por exemplo.

4.3 Considerações do Capítulo

Na pesquisa feita e dentre os trabalhos encontrados - (LINCKE, LUNDBERG e LÖWE, 2008), (KULKARNI, KALSHETTY e ARDE, 2010), (WEI e SALLIE, 1993), (COLEMAN, LOWTHER, *et al.*, 1994), (ZHOU e LEUNG, 2006), (WANG, HU, *et al.*, 2009) - o trabalho de Lincke *et al*. (2008), foi o que

[4] Conforme acesso realizado em 09 de junho de 2011 às 11h.

mais detalhadamente explorou as ferramentas para a coleta dos valores das métricas existentes à época de sua elaboração, já que, da lista inicial de ferramentas encontradas algumas, na época atual, já não estão mais disponíveis.

Outro trabalho, OO Metrics in Practice (DARCY e KEMERER, 2005), também explorou ferramentas para a coleta dos valores das métricas, restringindo-se a um conjunto comercial de ferramentas.

Ao final, a ferramenta “CKJM - Chidamber & Kemerer Java Metrics”, simples de usar, embora extremamente especializada no que se propõe, mostrou-se a mais adequada para este trabalho por não necessitar de ambiente específico, e nem da compilação do código-fonte, além de fornecer as seis métricas imediatamente à execução de um simples comando a partir de um arquivo do tipo texto.

5 PROCESSO DE MEDIÇÃO E ANÁLISE

O processo de medição prossegue, a partir da escolha da ferramenta, descrito no Capítulo 4, e baseia-se principalmente no trabalho de Kulkarni *et al.* (2010) que usa um procedimento bastante objetivo para a coleta e análise dos valores das métricas.

5.1 Introdução

Para que os procedimentos utilizados neste trabalho possam, por exemplo, ser reproduzidos, foi estruturado um conjunto de atividades que formam um processo genérico de medição. O processo está estruturado em cinco fases como apresentado na Figura 2. Os eventuais exemplos, cálculos e gráficos utilizados ou apresentados junto às atividades de cada fase do processo são apenas ilustrativos.

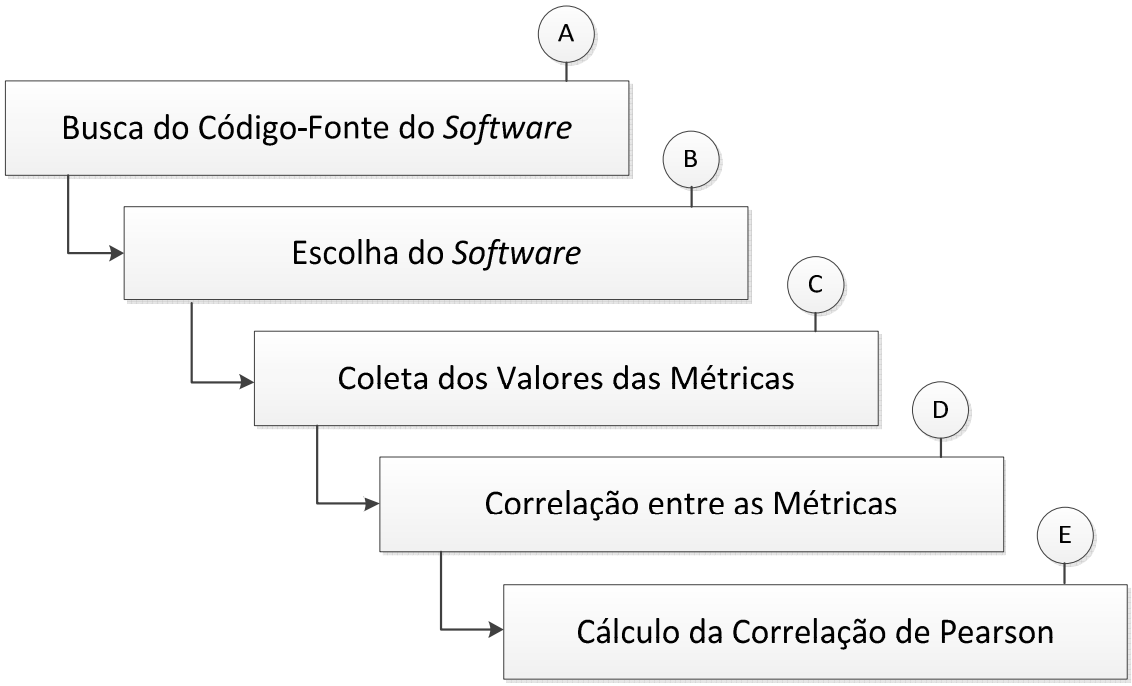

Figura 2: Diagrama do processo de medição onde estão indicadas as cinco fases necessárias para a coleta e correlacionamento das métricas.

O conjunto de atividades que formam o processo de medição prevê no passo A a execução de um conjunto de ações que direcione a busca do código-fonte pelos diversos repositórios disponíveis na internet. O detalhamento destas atividades está descrito no item 5.2.

Em seguida, o passo B do processo de medição prevê ações que orientem e, com base nos critérios descritos, indiquem a escolha do *software* cujo código-fonte será utilizado para a obtenção dos valores das métricas. O detalhamento deste passo está descrito no item 5.3.

O passo C, previsto no processo de medição, descreve a sequência de ações necessárias à obtenção das métricas, pelo uso da ferramenta já selecionada no item 4.2. O detalhamento deste passo está descrito no item 5.4.

O quarto passo do processo de medição, o passo D, descreve como correlacionar estatiscamente as métricas. O objetivo da verificação da existência de algum tipo de correlacionamento é verificar se existe correspondência, analogia ou interdependência entre as métricas de C&K. Se for identificado algum tipo de correlacionamento entre as métricas, poderá ajudar na análise da tendência da manutenibilidade. Há várias técnicas estatísticas de análise quantitativa que podem ser aplicadas e, neste trabalho, optou-se pela Correlação de Pearson em função de já ter sido referenciada em outros trabalhos relacionados com a manutenibilidade como o de Wang *et al* (WANG, HU, *et al.*, 2009), de Lincke *et al* (LINCKE, GUTZMANN e LÖWE, 2010) e o de Zhuo *et al* (ZHUO, LOWTHER, *et al.*, 1993). O detalhamento deste passo é apresentado no item 5.5.

A última fase do processo de medição, o passo E, apresenta uma amostra do resultado do cálculo da Correlação de Pearson e sua representação gráfica. O detalhamento deste passo é apresentado no item 5.6.

A hipótese a ser validada é o quanto a análise dos valores das métricas obtidos pode contribuir na avaliação da manutenibilidade do *software* em estudo, considerando as principais características do ambiente de desenvolvimento com base na orientação a objetos, isto é, herança, acoplamento e coesão.

5.2 Onde Buscar o Código-Fonte

A busca foi realizada em repositórios de *software* até que se encontrassem pacotes de código completos, em versões distintas e de programas que possam ter representatividade, medida através da quantidade

de *downloads* e do número de recomendações do *software* que acontecem no próprio repositório.

Há vários repositórios de *software* disponíveis na internet tais como github, Google Code, Assembla e Sourceforce. Optou-se, neste trabalho, pela busca no *SourceForge* (http://sourceforge.net/) devido aos seguintes fatores:

- É um repositório cuja filosofia de uso não envolve fins lucrativos;
- É um repositório onde se pode encontrar programas desenvolvidos sob licenciamento na forma de código aberto, entre outros, o que, nestes casos, permite acessar diretamente o código-fonte sem restrições na maior parte das vezes;
- Possui uma organização dos projetos que permite explorar não só as versões atuais dos *softwares* como também as versões anteriores, quando disponíveis;
- A maior parte dos 260.000[5] projetos existentes no *SourceForge* é escrito na linguagem Java.

Além da representatividade que o *software* pode ter, ou não, é preciso verificar se os valores das métricas obtidos junto às versões disponíveis possuem variedade suficiente de forma a permitir a avaliação das características relacionadas com:

- A complexidade estrutural do *software* (métricas WMC e RFC);
- A arquitetura do *software*, através da herança (métricas DIT e NOC);
- O acoplamento (métrica CBO);
- A coesão (métrica LCOM).

5.3 A Escolha do *Software*

No Capítulo 4 foi indicada a ferramenta (CKJM) a ser utilizada neste trabalho e o repositório onde serão procurados os *softwares* (*sourceforge.net*).

[5] Conforme informação disponível no sítio http://sourceforge.net/ obtida no acesso realizado em 09 de Junho de 2011 às 11h.

À medida que se explora o repositório nota-se que vários destes *softwares* não estão atualizados e outros possuem versões mais recentes. Optou-se por explorar o conjunto mais recente. O processo de busca inclui vasculhar as diversas versões do *software*, gravá-los no computador local e executar o procedimento para obtenção dos valores das métricas descrito no item 5.4. Portanto, mesmo sendo uma amostra para se avaliar o conteúdo, é necessário executar o procedimento para obtenção dos valores das métricas.

Considerando os critérios descritos, encontrou-se o Projeto jUnit, escrito integralmente na linguagem Java, comumente utilizado nas fases de testes durante o ciclo de desenvolvimento de *software*, contemplando 21 versões distintas, incluindo desde a versão 2.0 de 12/05/2003 até a versão 4.9b2 de 18/01/2011. A primeira versão disponível no repositório, a versão 2.0, contém 31 classes, enquanto a mais recente é formada por 563 classes.

5.4 Coletando os Valores das Métricas

Para cada versão de *software* disponível e que é avaliada, os dados são tabulados em planilhas eletrônicas de forma semelhante à que está apresentada na Tabela 7, envolvendo a identificação da versão do *software*, de cada uma das classes avaliadas e as correspondentes métricas C&K. A sequência de ações que compõe o procedimento necessário para a obtenção das métricas, utilizando a ferramenta "CKJM - Chidamber & Kemerer Java Metrics", é apresentado na Figura 3.

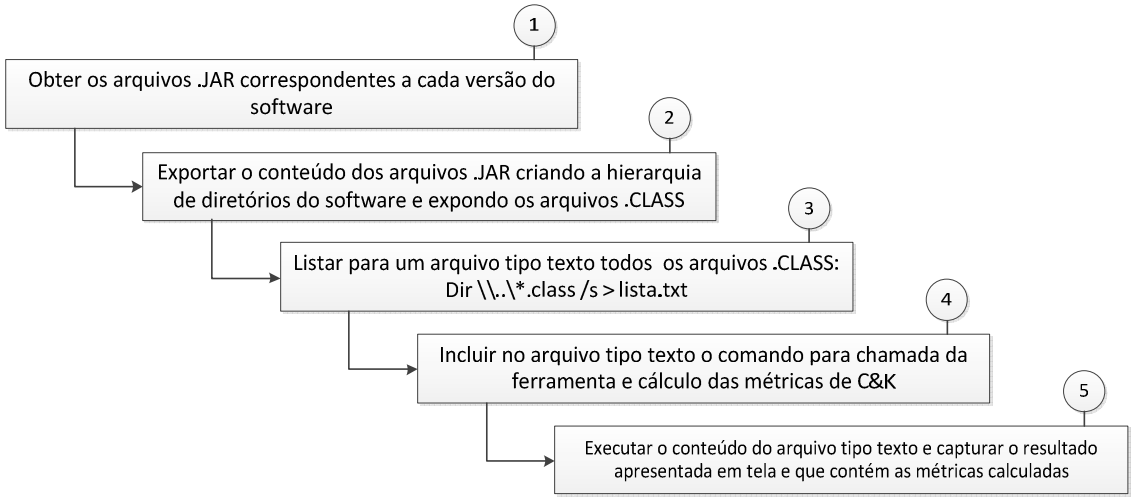

Figura 3: Procedimento para o uso da ferramenta CKJM que, ao final, fornece os valores de oito métricas incluindo as seis métricas de C&K.

Neste procedimento, o passo 1 é obter os arquivos .JAR, referentes à versão em avaliação, diretamente do repositório que, neste trabalho, é o *SourceForge*. O passo 2 é expandir estes arquivos para um diretório conhecido onde é criada a hierarquia de pastas e subpastas previstas na organização do código-fonte. Ao final deste passo o acesso aos arquivos .CLASS deve estar disponível.

O passo 3, executado em um ambiente Windows, lista todos os arquivos .CLASS para um arquivo tipo texto que ajudará a incluir os comandos da ferramenta CKJM no passo seguinte. Um exemplo deste comando é:

dir \\ ...\...*.class /s > \\...\...\lista.txt

O passo 4 contempla a inserção diretamente no texto do arquivo lista.txt, obtido ao final do passo 3, do comando que executa a chamada da ferramenta CKJM. A formatação detalhada deste comando pode ser obtida diretamente no sítio da ferramenta CKJM (http://www.spinellis.gr/sw/ckjm/). Um exemplo deste comando, em um ambiente Windows com Java VM 1.6, aplicado a título de exemplo à classe AllTests.class da versão 2.0 do *software* jUnit, é:

Java -Djava.ext.dir=C:\junit\junit2\junit2 -jar C:\CKJM\ckjm-1.8\build\ckjm-1.8.jar c:\junit\junit2\junit2\junit\samples\AllTests.class

Ao executar esta linha de comando a ferramenta CKJM fornece de imediato e na tela do computador o conjunto de valores das oito métricas, como abaixo:

junit.samples.AllTests 3 1 0 6 10 3 0 3

Os oito números à direita do nome da classe - junit.samples.AllTests - representam, respectivamente, os valores das seguintes métricas: WMC, DIT, NOC, CBO, RFC, LCOM, Ca[6] e NPM[7]. As duas últimas métricas são

[6] A métrica Ca – *Afferent Coupling*, mede o nível de responsabilidade desta classe, ou deste pacote, indicando o número de outros pacotes que dependem desta classe, ou das classes incluídas neste pacote. É uma medida relacionada ao acoplamento da classe.

[7] A métrica NPM – *Number of Public Methods for a Class*, foi proposta originalmente em *Object-Oriented Software Metrics* (LORENZ e KIDD, 1994) e reflete a quantidade de métodos declarados como públicos na classe em estudo. É uma medição relacionada ao acoplamento da classe e quanto maior seu valor, maior a complexidade da classe.

desconsideradas, neste trabalho, porque não compõem o conjunto das seis métricas de C&K. Portanto, a classe AllTests, na versão 2.0 do código do *software* jUnit apresenta as métricas de C&K indicadas na Tabela 7.

Tabela 7: Lista com as métricas de C&K para a classe AllTests na versão 2.0 do *software* jUnit.

Nome da Classe	Identificação das Métricas e exemplo de Valores					
	WMC	**DIT**	**NOC**	**CBO**	**RFC**	**LCOM**
junit.samples.AllTests	3	1	0	6	10	3

O procedimento se repete para cada classe, de cada versão do mesmo *software*. O resultado capturado diretamente da tela, ou copiado para outro arquivo tipo texto como, por exemplo, um arquivo XML, pode ser levado com facilidade para planilhas eletrônicas que ajudarão na tabulação de todos os valores das métricas e posterior análise comparativa. Executados os procedimentos, o resultado da compilação dos valores das métricas é apresentado na forma de tabela nos Anexos B, C e D.

5.5 Correlação entre as Métricas

A norma 1061 do IEEE (IEEE STD 1061, 2004, p. 11) faz recomendações para a aplicação de critérios de validação das métricas. Esta norma declara que uma métrica, para ser válida, deve ter um alto grau de associação com o atributo externo de qualidade e, quando esta associação está presente, a métrica pode ser usada como substituto do atributo de qualidade externo (IEEE STD 1061, 2004, p. 11).

Um dos critérios de validade da métrica citados na norma 1061 do IEEE é a correlação (IEEE STD 1061, 2004, p. 11). Por definição, correlação representa a interdependência de duas ou mais variáveis (INSTITUTO ANTÔNIO HOUAISS, 2009). Junto aos trabalhos pesquisados foram encontradas propostas para correlacionar variáveis, em particular os valores das métricas obtidos, através de alguma técnica estatística que investiga o grau de associação entre duas variáveis ou métricas. Há, na área estatística, várias formas de medição que podem ajudar a compreender a relação entre duas métricas. Uma delas denomina-se *Pearson Product-Moment – Correlation*

Coefficient, também conhecido, simplesmente, por *Pearson Correlation* ou Correlação de Pearson (PEARSON, 1892).

Esta correlação varia até +1, quando há uma forte correlação positiva entre duas variáveis que, neste trabalho são as métricas obtidas e associadas duas a duas, isto é, altos valores da primeira métrica estão associados a altos valores da segunda métrica. Simetricamente, baixos valores da primeira métrica estão associados a baixos valores da segunda métrica. A correlação varia até -1 quando há uma forte correlação negativa entre as duas métricas, isto é, altos valores da primeira métrica estão associados a baixos valores da segunda métrica e vice-versa. Quando o valor é exatamente zero, não há correlação entre as variáveis. A equação que representa a Correlação de Pearson é:

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)S_x S_y} \quad (1)$$

Onde:

r é o valor da Correlação de Pearson,

n é o número de ocorrências de cada variável,

$\boldsymbol{X_i}$ representa o valor individual de cada variável X,

$\boldsymbol{Y_i}$ representa o valor individual de cada variável Y,

$\boldsymbol{\bar{X}}$ representa o valor médio de todas as ocorrências da variável X,

$\boldsymbol{\bar{Y}}$ representa o valor médio de todas as ocorrências da variável Y,

$\boldsymbol{S_x}$ representa o desvio padrão computado a partir das vaiáveis X e

$\boldsymbol{S_y}$ representa o desvio padrão computado a partir das vaiáveis Y.

Neste trabalho o cálculo da Correlação de Pearson é feito pelo MS-Excel, integrante da suíte Microsoft Office Professional Plus 2010, versão 14.0.5128.5000 (64 bits), usando a seguinte função:

Pearson(matriz1;matriz2) (2)

Onde:

matriz1 representa a matriz formada pela linha ou coluna que contém os valores das variáveis X e

matriz2 representa a matriz formada pela linha ou coluna que contém os valores das variáveis Y.

Esta correlação foi utilizada no trabalho *Predicting Object-oriented Software Maintainability Using Multivariate Adaptive Regression Splines* (ZHOU e LEUNG, 2006), que propõe uma faixa de valores que caracterizaria a correlação entre as duas métricas avaliadas. Esta faixa de valores é apresentada na Tabela 8, para valores positivos da correlação, mas também pode se usada para os valores negativos da correlação.

Tabela 8: Faixas de valores que caracterizam a correlação entre duas métricas, propostas por Zhou *et* Leung - 2006.

Faixa de Variação da Correlação de Pearson (r)	Tipo de Correlação
r <= 0,1	insignificante
0,1 < r <= 0,3	pequena
0,3 < r <= 0,5	moderada
0,5 < r <= 0,7	grande
0,7 < r <= 0,9	muito grande
0.9 < r <= 1,0	quase perfeita

5.6 Cálculo da Correlação de Pearson

De acordo com a Tabela 16, no Anexo B, que reúne os valores das seis métricas de C&K obtidos a partir do conjunto de 31 classes da versão 2.0 do *software* jUnit, foram calculados os valores das Correlações de Pearson apresentados na Tabela 9, resultantes do agrupamento das métricas duas a duas.

Tabela 9: Conjunto de valores da Correlação de Pearson obtidos com base nas métricas geradas com as informações das 31 classes da versão 2.0 do *software* jUnit.

Par de Métricas	Correlação de Pearson	Tipo de Correlação
WMC – DIT	-0,03652	insignificante
WMC – CBO	0,567474	grande
WMC – RFC	0,944353	quase perfeita
WMC – LCOM	0,609024	grande
DIT – CBO	0,107093	pequena
DIT – RFC	0,018008	insignificante
DIT – LCOM	0,001756	insignificante
CBO – RFC	0,688212	grande
CBO – LCOM	0,719398	muito grande
RFC - LCOM	0,606733	grande

O cálculo final não considera a ordem adotada, ou seja, o valor da Correlação de Pearson quando considerado o par de variáveis X e Y é o mesmo valor obtido quando é considerado o par de variáveis Y e X.

Calculadas as Correlações de Pearson para todas as versões do *software* jUnit, os resultados estão compilados na Tabela 15, no Anexo A.

Em busca de uma melhor forma de visualizar os valores da Correlação de Pearson, sua transposição para um conjunto de gráficos pode ajudar na análise dos resultados obtidos, apresentados parcialmente no Gráfico 1.

5.7 Considerações do Capítulo

O procedimento adotado não é complexo embora trabalhoso à medida que o *software* tem sua quantidade de classes aumentada. Porém, mostrou-se estável e isento de erros durante o processamento, podendo ser executado múltiplas vezes desde que os arquivos intermediários sejam preservados.

O resultado obtido mostrou-se válido e correto a partir do momento que foi confrontado com os valores das mesmas métricas obtidos manualmente, por amostragem.

6 EXPERIMENTO E ANÁLISE DOS DADOS

Neste capítulo são apresentados o ambiente utilizado e os resultados do experimento voltado para cada métrica de C&K. Em seguida, é feita análise dos dados obtidos.

6.1 Introdução

Inicia-se mostrado como o ambiente foi montado, como a ferramenta foi instalada e as principais dificuldades.

É possível que os ciclos de coleta de métricas necessitem ser processados múltiplas vezes. Assim, criados os ambientes, eles são preservados para reutilização, caso necessário.

A ferramenta selecionada atuará em cada versão do *software* gerando um conjunto de métricas que são tabuladas em planilhas eletrônicas.

Realizada a tabulação dos dados, em seguida é feita a análise estatística dos dados para avaliar a manutenibilidade das versões do *software* em estudo.

6.2 Montagem do Ambiente

O ambiente montado para a obtenção dos valores das métricas é formado pelos seguintes itens:

- Microcomputador com 4GB de memória RAM e 500 GB de disco rígido e processador Intel Core 2 DUO P7450 de 2,13GHz;
- Sistema operacional Windows 7 - 64bits e *Service Pack* 1, com instalação padrão;
- *Microsoft Office Professional Plus 2010* versão 14.0.6023.1000 64bits, para edição de planilhas eletrônicas, com instalação padrão;
- *Java Machine Version 6 – Update 26* como interpretador da linguagem Java, com instalação padrão;
- *Software* CKJM versão 1.8 escolhida como a ferramenta para a obtenção dos valores das métricas. Para instala-la basta expandir o arquivo correspondente que está compactado;

- Notepad para a edição de textos UNICODE. Integra a instalação do Windows 7, não necessitando de instalação adicional;
- Pacotes com os códigos-fonte das versões do *software* jUnit, retirados do sítio http://sourceforge.net/projects/junit/ .

A montagem do ambiente não requer sofisticação ou técnica especializada. Assim como o uso da ferramenta CKJM, conforme descrito no item 5.4.

6.3 Experimento e Análise dos Dados - WMC

Com base nos valores da métrica WMC ao longo das 21 diferentes versões do *software* jUnit, observa-se que a distribuição de seus valores está dentro do que é sugerido, como apresentado na Tabela 10. As faixas de valores, indicados em porcentagem, são as mesmas utilizadas em Kulkarni *et al.* (KULKARNI, KALSHETTY e ARDE, 2010, p. 647).

Tabela 10: Tabela com o agrupamento dos valores da métrica WMC indicados em porcentagem, de acordo com a faixa de valores obtidos em cada versão do *software* jUnit.

Versão do *software* jUnit	Quantidade Total de Classes da Versão	Valores da Métrica WMC em Porcentagem			
		0 <= WMC < 10	11 <= WMC < 20	21 <= WMC <= 30	WMC > 30
2.0	31	87,10%	9,68%	3,23%	0,00%
2.1	36	88,89%	5,56%	5,56%	0,00%
3.0	57	91,23%	5,26%	3,51%	0,00%
3.2	57	91,23%	5,26%	3,51%	0,00%
3.4	50	86,00%	12,00%	2,00%	0,00%
3.5	59	88,14%	10,17%	1,69%	0,00%
3.6	59	88,14%	10,17%	1,69%	0,00%
3.7	59	88,14%	10,17%	1,69%	0,00%
3.8	79	92,41%	5,06%	1,27%	1,27%
3.8.1	81	90,12%	8,64%	1,23%	0,00%
3.8.2	87	89,66%	9,20%	1,15%	0,00%
4.0	240	93,33%	4,17%	2,08%	0,42%
4.1	246	93,50%	4,07%	2,03%	0,41%
4.2	257	93,77%	3,89%	1,95%	0,39%
4.3.1	267	94,01%	3,75%	1,87%	0,37%
4.4	309	93,53%	4,21%	1,94%	0,32%

Tabela 10 (continuação): Tabela com o agrupamento dos valores da métrica WMC indicados em porcentagem, de acordo com a faixa de valores obtidos em cada versão do *software* jUnit.

Versão do *software* jUnit	Quantidade Total de Classes da Versão	Valores da Métrica WMC em Porcentagem			
		0 <= WMC < 10	11 <= WMC < 20	21 <= WMC <= 30	WMC > 30
4.5	405	94,81%	3,46%	1,48%	0,25%
4.6	431	94,90%	3,48%	1,39%	0,23%
4.7	482	95,02%	3,32%	1,45%	0,21%
4.8	508	95,08%	3,35%	1,38%	0,20%
4.9b2	563	95,38%	3,37%	1,07%	0,18%

O que se observa na Tabela 10, é que acima de 87% das classes de todas as versões do *software* jUnit atendem às recomendações sugeridas quanto aos valores admissíveis da métrica WMC, cuja variação deve estar entre 1 e 10 (KULKARNI, KALSHETTY e ARDE, 2010, p. 647). Verifica-se, também, que entre as versões 3.8 e 4.0 houve uma reestruturação interna do *software*, que se pôde confirmar, posteriormente, por seus desenvolvedores, através de mensagens eletrônicas trocadas com Kent Beck (kent@threeriversinstitute.org). Este fato é real devido à mudança na quantidade de classes do *software* jUnit que passou de 79, na versão 3.8, para 240 na versão 4.0. A versão 3.8 foi a primeira a apresentar uma classe - junit.tests.framework.AssertTest – com WMC maior que 30.

A partir da versão 4.0 do *software* jUnit, observa-se que a porcentagem dos valores medidos da métrica WMC com valor menor ou igual a 10 cresceu mantendo-se, a partir daí, acima de 93%. Quanto maior o valor de WMC maior o número de métodos, mais complexa é a classe, maior é a dificuldade de entendimento da classe e mais difícil é sua manutenibilidade (WEI e SALLIE, 1993, p. 113).

Quando comparada às demais métricas, através da Correlação de Pearson que se observa no Gráfico 1, verifica-se que, ao longo das versões do *software* jUnit, as linhas nos diagramas permanecem com pouca variação e, quando esta variação ocorre está em torno da versão 4.0, mas estabiliza-se nas versões seguintes.

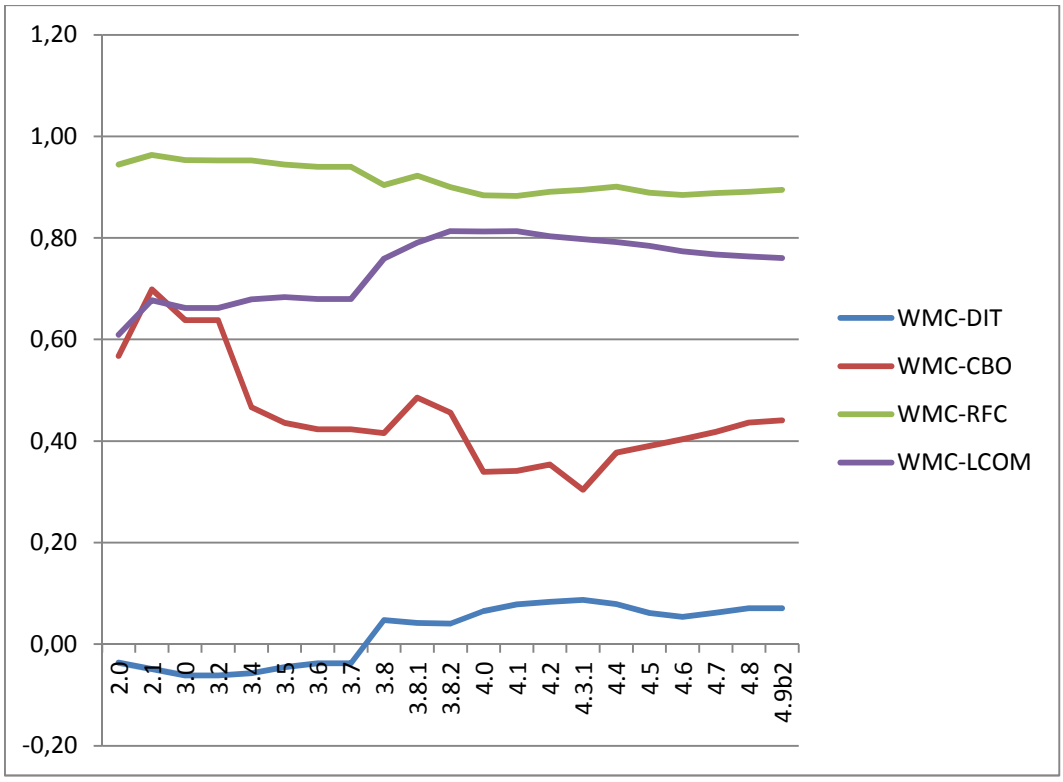

Gráfico 1: Gráfico representativo das Correlações de Pearson obtidas com base nas métricas calculadas pela ferramenta CKJM considerando as 21 versões do *software* jUnit, enfatizando o agrupamento das métricas que envolvem WMC.

O efeito entre WMC e DIT, no Gráfico 1 em azul, não é relevante porque, ao longo das diversas versões, a Correlação de Pearson nunca foi maior que 0,1 , ou seja, uma correlação insignificante, e nunca menor que -0,1 , também uma correlação insignificante de acordo com a classificação da Tabela 8.

Os valores da Correlação de Pearson entre WMC e CBO, no Gráfico 1 em vermelho, indicam que nas duas primeiras versões os valores da métrica de acoplamento (CBO) aumentaram à medida que o WMC também aumentou. A partir da versão 3.0, o CBO passou a diminuir, sofrendo pequena turbulência entre as versões 3.8 e 4.0, diminuindo um pouco mais e estabilizando-se com valores em torno de 0,4, ou seja, uma correlação moderada de acordo com a classificação da Tabela 8.

Quase que simultaneamente, a correlação entre os valores da métrica que mede a falta de coesão (LCOM) e o WMC, no Gráfico 1 em lilás, teve comportamento inverso, estabilizando-se próximo de 0,8, ou seja, uma correlação muito grande de acordo com a classificação da Tabela 8, denunciando que a diferença entre o número de pares de métodos das classes

de cada versão sem variáveis compartilhadas e o número de pares de métodos com variáveis compartilhadas permanece estável.

Por último a correlação entre a métrica que se propõe aferir a comunicação entre objetos (RFC) e o WMC, no Gráfico 1 em verde, manteve-se quase sempre acima de 0,9, ou seja, uma correlação quase perfeita de acordo com a classificação da Tabela 8, significando que o aumento do valor do WMC provoca um aumento no valor do RFC.

As demais faixas de valores da métrica WMC que se observa na Tabela 10, somam valores, em porcentagem, cada vez menores ao longo das versões do *software* jUnit, reforçando as indicações de que a manutenibilidade tende a ser melhor já que a quantidade de classes com valores de WMC acima de 10 diminui ao longo das versões.

6.4 Experimento e Análise dos Dados - RFC

Com base nos valores da métrica RFC ao longo das 21 diferentes versões do *software* jUnit, observa-se que a distribuição, na forma de porcentagem, dos valores de RFC na faixa compreendida até 20 cresce desde a primeira versão e, na mais recente, a versão 4.9b2, está acima de 92%, como é apresentado na Tabela 11. As faixas de valores, indicados em porcentagem, são as mesmas utilizadas em Kulkarni *et al.* (KULKARNI, KALSHETTY e ARDE, 2010, p. 648).

A métrica RFC, integrante do conjunto proposto por C&K representa a soma dos métodos locais com os métodos chamados pelos métodos locais. Como se observa na Tabela 11, ao longo das consecutivas versões a porcentagem das classes que possuem o valor do RFC menor que 20 cresce ao longo das versões, enquanto que, de uma forma geral, os números nas demais faixas diminuem. À luz dos valores da Tabela 11 nota-se que a manutenibilidade do *software* jUnit melhora ao longo das consecutivas versões já que quanto menor a quantidade de métodos envolvidos com o valor do RFC mais fácil é a manutenção da classe (WEI e SALLIE, 1993, p. 113).

O inverso, ou seja, um valor maior de RFC, embora possa representar uma maior capacidade de resposta da classe, exige estrutura adequada,

promovendo uma maior complexidade da classe, o que exige maior esforço de teste (BORGES, 2006, p. 28).

Tabela 11: Tabela com o agrupamento dos valores da métrica RFC indicados em porcentagem, de acordo com a faixa de valores obtidos em cada versão do *software* jUnit.

Versão do *software* jUnit	Quantidade Total de Classes da Versão	Valores da Métrica RFC em Porcentagem				
		0<= RFC <=20	21<= RFC <=40	41<= RFC <=60	61<= RFC <=80	RFC>80
2.0	31	77,42%	16,13%	6,45%	0,00%	0,00%
2.1	36	80,56%	11,11%	8,33%	0,00%	0,00%
3.0	57	84,21%	10,53%	5,26%	0,00%	0,00%
3.2	57	84,21%	10,53%	5,26%	0,00%	0,00%
3.4	50	82,00%	14,00%	4,00%	0,00%	0,00%
3.5	59	84,75%	11,86%	3,39%	0,00%	0,00%
3.6	59	84,75%	11,86%	3,39%	0,00%	0,00%
3.7	59	84,75%	11,86%	3,39%	0,00%	0,00%
3.8	79	87,34%	7,59%	5,06%	0,00%	0,00%
3.8.1	81	86,42%	9,88%	3,70%	0,00%	0,00%
3.8.2	87	85,06%	11,49%	3,45%	0,00%	0,00%
4.0	240	88,33%	8,75%	2,92%	0,00%	0,00%
4.1	246	88,21%	8,94%	2,85%	0,00%	0,00%
4.2	257	88,72%	8,17%	2,72%	0,39%	0,00%
4.3.1	267	89,89%	7,12%	2,62%	0,00%	0,37%
4.4	309	89,97%	6,80%	2,91%	0,00%	0,32%
4.5	405	90,62%	6,67%	2,47%	0,00%	0,25%
4.6	431	90,72%	6,50%	2,32%	0,23%	0,23%
4.7	482	91,29%	6,22%	2,07%	0,21%	0,21%
4.8	508	91,54%	6,10%	1,97%	0,20%	0,20%
4.9b2	563	92,18%	5,68%	1,78%	0,18%	0,18%

Quando se compara a correlação entre os valores da métrica RFC e as demais, como é apresentado no Gráfico 2, nota-se um valor próximo de 0,8 para a correlação entre RFC, WMC e CBO, isto é, uma correlação muito grande de acordo com a classificação da Tabela 8. Este nível de valor recomenda a dedicação de recursos extras para testes da classe ou componente (CHIDAMBER, DARCY e KEMERER, 1998, p. 635). A correlação entre RFC e DIT não demonstra influência porque seus valores, quando

positivos, alcançam 0,1, ou seja, uma correlação insignificante de acordo com a classificação da Tabela 8.

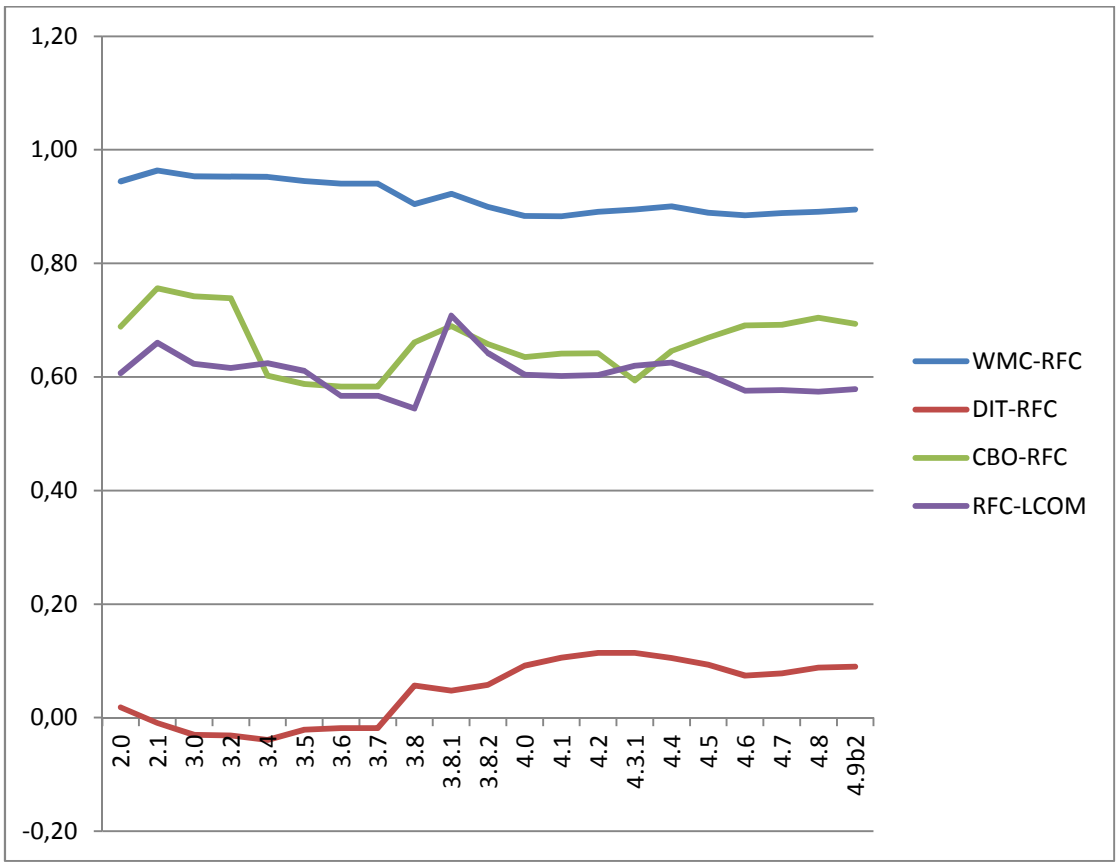

Gráfico 2: Gráfico representativo das Correlações de Pearson obtidas com base nas métricas calculadas pela ferramenta CKJM considerando as 21 versões do *software* jUnit, enfatizando o agrupamento das métricas que envolvem RFC.

6.5 Experimento e Análise dos Dados - CBO

Com base nos valores da métrica CBO ao longo das 21 diferentes versões do *software* jUnit, observa-se que a distribuição, na forma de porcentagem, dos valores de CBO na faixa compreendida até 5 se manteve, na média, acima de 87%, como pode ser calculado a partir da Tabela 12. As faixas de valores, indicados em porcentagem, são as mesmas utilizadas em Kulkarni *et al.* (KULKARNI, KALSHETTY e ARDE, 2010, p. 648).

Como o CBO é uma medição do acoplamento entre classes, ou seja, o quanto determinada classe usa métodos ou atributos de outra classe e vice-versa, quando são somados, em porcentagem, os valores do CBO até 10, a média alcança 96,99%. Valores baixos de CBO significam que a classe se acopla a poucas outras classes, o que favorece sua compreensibilidade e

reusabilidade (KULKARNI, KALSHETTY e ARDE, 2010, p. 649). Bons projetos de *software* obedecem ao princípio de baixo acoplamento (BRIAND, DEVANBU e MELO, 1997, p. 412), enquanto que o alto acoplamento tende a introduzir falhas causadas pelas ligações entre as classes (TANG, KAO e CHEN, 1999, p. 4).

Tabela 12: Tabela com o agrupamento dos valores da métrica CBO, em porcentagem, de acordo com a faixa de valores obtidos em cada versão do *software* jUnit.

Versão do *software* jUnit	Quantidade Total de Classes da Versão	Valores da Métrica CBO em Porcentagem				
		0 <= CBO < 5	5 <= CBO < 10	10 <= CBO < 15	15 <= CBO < 20	CBO >= 20
2.0	31	87,10%	9,68%	3,23%	0,00%	0,00%
2.1	36	77,78%	19,44%	0,00%	2,78%	0,00%
3.0	57	89,47%	7,02%	1,75%	0,00%	1,75%
3.2	57	89,47%	7,02%	1,75%	0,00%	1,75%
3.4	50	88,00%	8,00%	4,00%	0,00%	0,00%
3.5	59	88,14%	8,47%	3,39%	0,00%	0,00%
3.6	59	88,14%	8,47%	3,39%	0,00%	0,00%
3.7	59	88,14%	8,47%	3,39%	0,00%	0,00%
3.8	79	83,54%	12,66%	2,53%	1,27%	0,00%
3.8.1	81	83,95%	12,35%	2,47%	1,23%	0,00%
3.8.2	87	86,21%	10,34%	2,30%	1,15%	0,00%
4.0	240	88,33%	9,58%	1,67%	0,42%	0,00%
4.1	246	88,21%	9,76%	1,63%	0,41%	0,00%
4.2	257	88,72%	9,34%	1,56%	0,39%	0,00%
4.3.1	267	89,89%	8,24%	1,50%	0,37%	0,00%
4.4	309	89,64%	8,09%	1,94%	0,32%	0,00%
4.5	405	88,64%	8,15%	2,96%	0,25%	0,00%
4.6	431	87,70%	9,05%	2,78%	0,46%	0,00%
4.7	482	88,80%	8,30%	2,49%	0,41%	0,00%
4.8	508	89,17%	7,87%	2,56%	0,39%	0,00%
4.9b2	563	89,52%	7,99%	2,13%	0,36%	0,00%

À medida que os valores do CBO crescem, maior é o acoplamento da classe, pode haver um comprometimento do encapsulamento da classe, a oportunidade de seu reuso através de herança diminui, maiores esforços no processo de testes da classe são exigidos e a dificuldade em entender a estrutura da classe também aumenta, o que dificulta sua manutenibilidade

(CHIDAMBER, DARCY e KEMERER, 1998, p. 631), (HARRISON, COUNSELL e NITHI, 1998, p. 6).

Quando se observam os valores da Correlação de Pearson envolvendo CBO, ao longo das consecutivas versões, como é apresentado no Gráfico 3, notam-se variações intensas na correlação entre CBO-WMC e CBO-LCOM, particularmente em torno da versão 4.0. A correlação CBO-WMC cai de um valor próximo de 0,5, classificada como correlação grande de acordo com a Tabela 8, na versão 3.8 para um valor próximo de 0,3 na versão 4.2, classificada como correlação pequena.

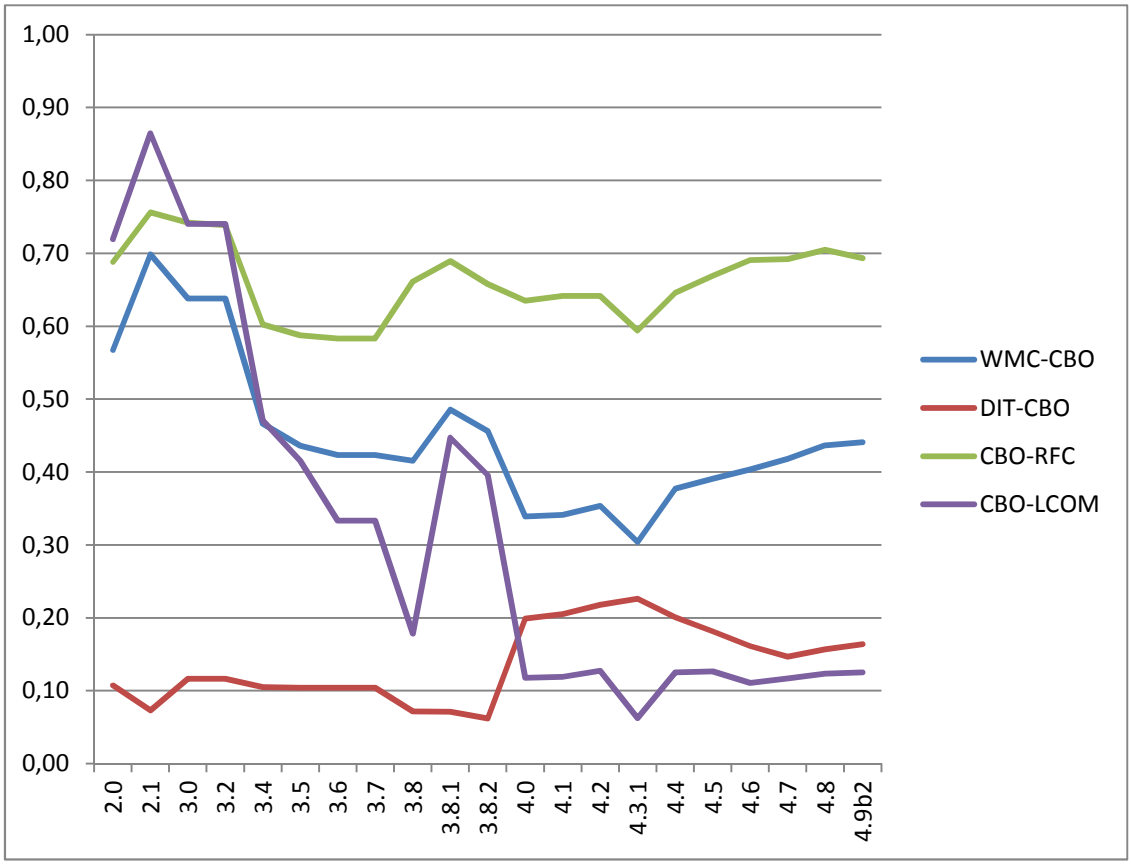

Gráfico 3: Gráfico representativo das Correlações de Pearson obtidas com base nas métricas calculadas pela ferramenta CKJM considerando as 21 versões do *software* jUnit, enfatizando o agrupamento das métricas que envolvem CBO.

Devido ao valor da correlação ser positivo, à medida que uma métrica cresce a outra também cresce, porém devido à diminuição da correlação para 0,3 significa que o crescimento é proporcionalmente menor. Indicando que se houver um crescimento na quantidade de métodos (WMC) o acoplamento não

crescerá na mesma proporção fazendo com que a piora da manutenibilidade também seja influenciada em uma proporção menor.

6.6 Experimento e Análise dos Dados - LCOM

Assim como a métrica RFC é uma medição da complexidade da classe devido ao acoplamento, LCOM também é uma medição da complexidade da classe derivada da coesão (ZHOU e LEUNG, 2006, p. 7).

Com base nos valores da métrica LCOM ao longo das 21 diferentes versões do *software* jUnit, observa-se que a distribuição, na forma de porcentagem, dos valores de LCOM na faixa compreendida até 20 se manteve, na média, em 90,67%, como pode ser calculado a partir da Tabela 13.

Tabela 13: Tabela com o agrupamento dos valores da métrica LCOM indicados em porcentagem, de acordo com a faixa de valores obtidos em cada versão do *software* jUnit.

Versão do *software* jUnit	Quantidade Total de Classes da Versão	Valores da Métrica LCOM em Porcentagem				
		0 <= LCOM < 20	20 <= LCOM < 40	40 <= LCOM < 60	60 <= LCOM <80	LCOM >= 80
2.0	31	87,10%	6,45%	0,00%	3,23%	3,23%
2.1	36	88,89%	2,78%	2,78%	2,78%	2,78%
3.0	57	92,98%	3,51%	0,00%	1,75%	1,75%
3.2	57	92,98%	3,51%	0,00%	1,75%	1,75%
3.4	50	88,00%	4,00%	2,00%	4,00%	2,00%
3.5	59	89,83%	3,39%	1,69%	3,39%	1,69%
3.6	59	89,83%	3,39%	1,69%	1,69%	3,39%
3.7	59	89,83%	3,39%	1,69%	1,69%	3,39%
3.8	79	92,41%	2,53%	1,27%	0,00%	3,80%
3.8.1	81	88,89%	3,70%	2,47%	1,23%	3,70%
3.8.2	87	87,36%	4,60%	2,30%	0,00%	5,75%
4.0	240	90,83%	3,75%	1,67%	0,00%	3,75%
4.1	246	91,06%	3,66%	1,63%	0,00%	3,66%
4.2	257	91,44%	3,50%	0,78%	0,39%	3,89%
4.3.1	267	91,76%	3,00%	1,50%	0,00%	3,75%
4.4	309	90,29%	4,21%	1,29%	0,32%	3,88%
4.5	405	91,36%	3,95%	1,23%	0,49%	2,96%
4.6	431	91,88%	3,71%	1,16%	0,46%	2,78%
4.7	482	92,12%	3,53%	1,04%	0,62%	2,70%
4.8	508	92,32%	3,35%	0,98%	0,59%	2,76%
4.9b2	563	92,90%	3,02%	0,89%	0,53%	2,66%

As faixas de valores, indicados em porcentagem na Tabela 13, são as mesmas utilizadas em Kulkarni *et al.* (KULKARNI, KALSHETTY e ARDE, 2010, p. 648).

Classes com alto valor de LCOM são difíceis de serem implementadas (CHIDAMBER, DARCY e KEMERER, 1998, p. 1), tornando os desenvolvedores menos produtivos, requerendo maior tempo para implantação por linha de código (CHIDAMBER, DARCY e KEMERER, 1998, p. 8). Tais classes, com alto LCOM são mais propensas a erros e exigem maiores esforços em relação aos seus testes (CHIDAMBER, DARCY e KEMERER, 1998, p. 8). Ainda pelo alto valor de LCOM, o comportamento destas classes pode ter menor previsibilidade, influenciando sua manutenibilidade de forma negativa.

Altos valores de LCOM podem ser encontrados no início do desenvolvimento dos projetos, quando a compreensão do funcionamento da classe ainda não está totalmente madura (CHIDAMBER, DARCY e KEMERER, 1998, p. 8). Verifica-se que é esta a situação. O valor de LCOM abaixo de 20, em porcentagem, na versão inicial é 87,10%, ou seja, esta é a porcentagem de classes com valor de LCOM menor que 20. À medida que novas versões consecutivas do *software* jUnit são produzidas, observa-se que o mesmo valor de LCOM, menor que 20, em porcentagem, sobe para 92,90% na versão mais recente, representando uma melhoria na coesão destas classes e, portanto em sua manutenibilidade (WEI e SALLIE, 1993, p. 3).

Estes cenários com altos valores de LCOM estão indicados também pela Correlação de Pearson apresentada no Gráfico 4. Neste se observa variação intensa na correlação que envolve CBO e LCOM. Nas duas versões iniciais do *software* jUnit os valores da correlação estavam em torno de 0,8, ou seja, uma correlação classificada como muito grande, de acordo com a Tabela 8.

Esta correlação transformou-se em uma correlação pequena baixando para perto de 0,2 na versão 3.8. Sofreu turbulências desta versão até a versão 4.0 e permaneceu como uma correlação pequena com valor em torno de 0,1 até a versão mais recente. Esta correlação, quando em torno de 0,8 significa que quando a métrica CBO aumenta, a falta de coesão – LCOM, também aumenta

o que influencia negativamente o desenvolvimento do código (CHIDAMBER, DARCY e KEMERER, 1998, p. 8) e sua manutenibilidade.

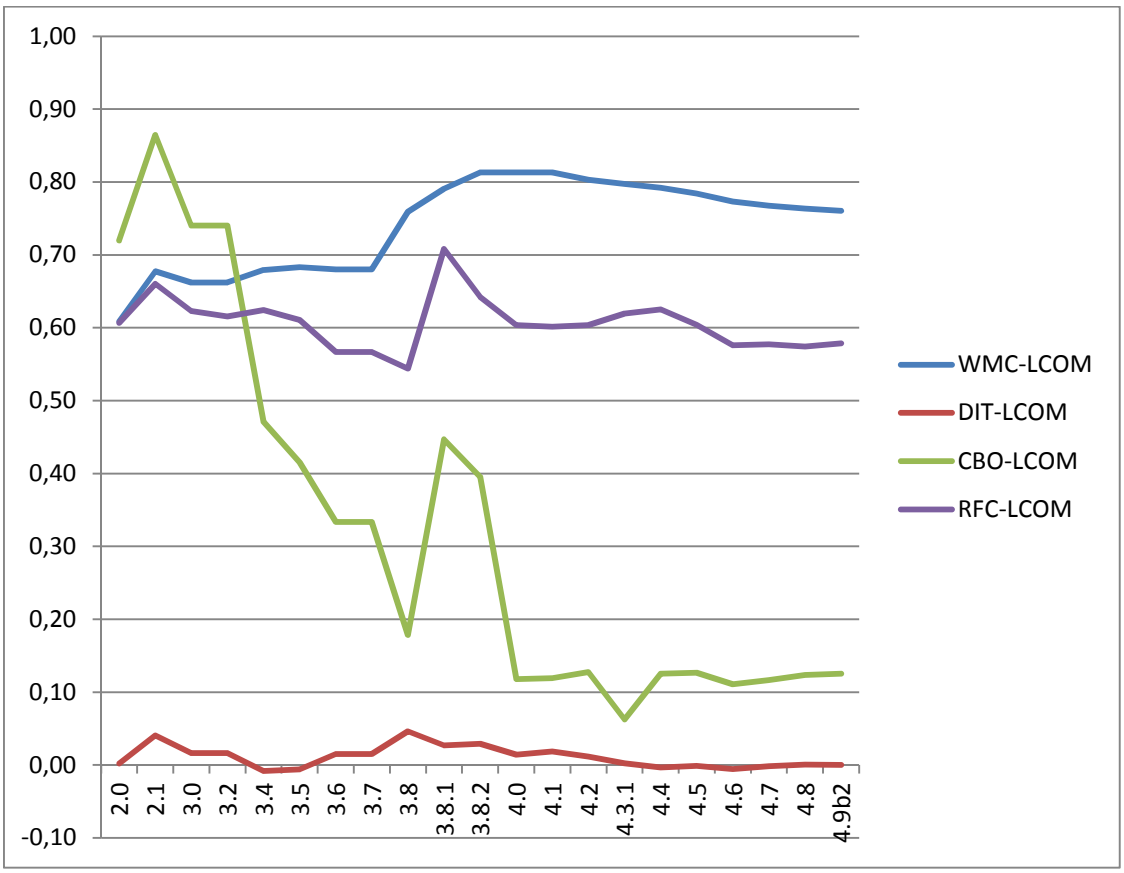

Gráfico 4: Gráfico representativo das Correlações de Pearson obtidas com base nas métricas calculadas pela ferramenta CKJM considerando as 21 versões do *software* jUnit, enfatizando o agrupamento das métricas que envolvem LCOM.

6.7 Experimento e Análise dos Dados - DIT

Observa-se ao longo das 21 versões consecutivas do *software* jUnit que a herança entre classes é pouco explorada. Na Tabela 14, é apresentado, em porcentagem, o conjunto de valores da métrica DIT. As faixas de valores são as mesmas utilizadas em Kulkarni *et al.* (KULKARNI, KALSHETTY e ARDE, 2010, p. 649).

Nas duas versões iniciais, em torno de 70% das classes tinham DIT igual a dois. Nas nove versões seguintes, até a 3.8.2, mais de 60% das classes tinha DIT igual a três. A partir da versão 4.0 e até a versão mais recente, a porcentagem de classes com DIT igual a um predomina, inicialmente com

aproximadamente 57%, chegando próximo de 70% na versão mais recente. A métrica DIT indica a complexidade da classe relacionada com a herança.

Recomenda-se trabalhar com um alto valor de DIT e um baixo valor para a métrica NOC (KULKARNI, KALSHETTY e ARDE, 2010, p. 647).

Tabela 14: Tabela com o agrupamento dos valores da métrica DIT indicados em porcentagem, de acordo com a faixa de valores obtidos em cada versão do *software* jUnit.

Versão do *software* jUnit	Quantidade Total de Classes da Versão	Valores da Métrica DIT em Porcentagem					
		DIT=0	DIT=1	DIT=2	DIT=3	DIT=4	DIT=5
2.0	31	6,45%	22,58%	70,97%	0,00%	0,00%	0,00%
2.1	36	5,56%	19,44%	75,00%	0,00%	0,00%	0,00%
3.0	57	8,77%	12,28%	5,26%	64,91%	8,77%	0,00%
3.2	57	8,77%	12,28%	5,26%	64,91%	8,77%	0,00%
3.4	50	10,00%	12,00%	4,00%	64,00%	10,00%	0,00%
3.5	59	8,47%	15,25%	3,39%	64,41%	8,47%	0,00%
3.6	59	8,47%	15,25%	3,39%	64,41%	8,47%	0,00%
3.7	59	8,47%	15,25%	3,39%	64,41%	8,47%	0,00%
3.8	79	10,13%	16,46%	6,33%	60,76%	6,33%	0,00%
3.8.1	81	9,88%	16,05%	6,17%	61,73%	6,17%	0,00%
3.8.2	87	9,20%	16,09%	6,90%	62,07%	5,75%	0,00%
4.0	240	7,92%	57,92%	5,42%	27,50%	1,25%	0,00%
4.1	246	8,13%	58,13%	6,10%	26,83%	0,81%	0,00%
4.2	257	8,17%	59,53%	5,84%	25,68%	0,78%	0,00%
4.3.1	267	7,87%	61,05%	5,99%	24,34%	0,75%	0,00%
4.4	309	6,80%	63,11%	8,41%	21,04%	0,65%	0,00%
4.5	405	6,17%	67,65%	8,15%	16,30%	1,23%	0,49%
4.6	431	5,80%	66,82%	8,58%	16,94%	1,39%	0,46%
4.7	482	5,81%	68,88%	8,51%	15,35%	1,04%	0,41%
4.8	508	5,71%	70,28%	8,07%	14,57%	0,98%	0,39%
4.9b2	563	5,86%	69,63%	8,70%	14,39%	1,07%	0,36%

Como os valores da métrica DIT eram maiores nas versões iniciais do *software* jUnit isto sugere que a manutenibilidade das versões iniciais era favorecida, comparativamente com as versões mais recentes.

A herança, no paradigma de orientação a objeto, faz com que a classe se acople com suas classes ascendentes e descendentes (BRIAND, DEVANBU e MELO, 1997, p. 412), sugerindo um aumento do acoplamento da classe o que

se contrapõe ao princípio que recomenda baixo acoplamento para um bom projeto de *software* (BRIAND, DEVANBU e MELO, 1997, p. 412).

Quando se avalia a Correlação de Pearson entre a métrica DIT e as demais, através do Gráfico 5, nota-se que a maior parte de seus valores está próximo de 0,1 nas versões mais recentes, o que classifica a correlação como insignificante, conforme a Tabela 8.

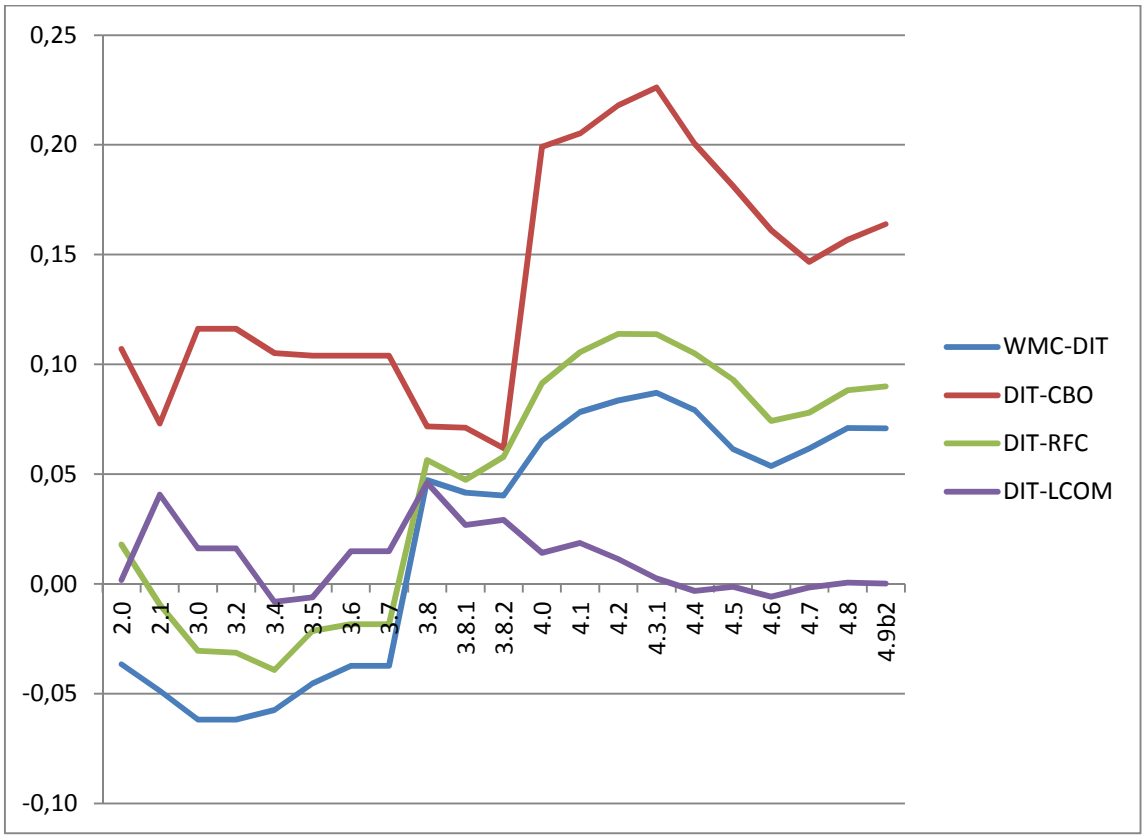

Gráfico 5: Gráfico representativo das Correlações de Pearson obtidas com base nas métricas calculadas pela ferramenta CKJM considerando as 21 versões do *software* jUnit, enfatizando o agrupamento das métricas que envolvem DIT.

Desta forma, é possível afirmar que estatiscamente a correlação entre a métrica DIT e as demais métricas não permite avaliar a manutenibilidade das versões do *software* jUnit.

6.8 Experimento e Análise dos Dados - NOC

Os valores da métrica NOC são iguais a zero para as classes das versões do *software* jUnit, com apenas uma exceção que indica uma classe na versão 4.9b2 que possui NOC igual a um.

Sendo a predominante a maioria dos valores da métrica NOC igual a zero, não foi possível considera-la na comparação com as demais métricas e nem relaciona-la com a manutenibilidade das versões do *software* jUnit.

6.9 Considerações do Capítulo

A observação dos dados e dos Gráficos 1, 2, 3, 4, e 5, permite os seguintes comentários:

a) Das seis métricas de C&K, uma delas – NOC, não aparece. O motivo é que das 3800 classes envolvidas em todas as medições, apenas uma classe, na versão 4.9b2 possui NOC diferente de zero e, neste caso, é igual a um, indicando que somente esta classe possui uma classe filha, permitindo desconsiderar esta métrica no processo de análise;

b) A métrica DIT apresenta a Correlação de Pearson menor que 0,1, na maioria dos cálculos, e pouco acima de 0,1 em uma quantidade pequena dos valores, indicando que na correlação entre esta métrica e as demais, existe um relacionamento praticamente insignificante;

c) Há variações bruscas da Correlação de Pearson entre as versões 3.8, 3.8.1, 3.8.2 e 4.0, justamente quando o software passou de um patamar de 79 classes que o compunham na versão 3.8, para 240 classes na versão 4.0 mostrando que, ao longo destas versões, houve uma provável mudança de arquitetura e a coesão representada pelas métricas CBO e LCOM passaram por flutuações importantes.

O experimento aqui registrado usa o conjunto de métricas de C&K, que retratam o comportamento estático do *software*, como ferramenta para avaliar a manutenibilidade do *software* jUnit ao longo de várias e consecutivas versões do mesmo código-fonte. Após a análise estatística este conjunto de métricas pode mostrar, em relação à manutenibilidade, os possíveis problemas e uma ideia do que deve ser feito, mas não onde estão os problemas e nem a solução que deve ser adotada (DASKALANTONAKIS, 1992, p. 1009).

Cabe lembrar que os resultados aqui obtidos são válidos para este experimento, não significando que para outros projetos em Java e com base nas métricas de C&K, os resultados serão semelhantes.

7 CONCLUSÃO E TRABALHOS FUTUROS

Neste capítulo é apresentada a conclusão alcançada com a ajuda do experimento que envolveu a análise estatística dos valores das métricas obtidos, bem como sugestões para temas a serem explorados em trabalhos futuros.

7.1 Contribuições do Trabalho

Tendo em mãos um conjunto de valores das métricas propostas por C&K, que se relacionam diretamente, conforme suas definições, a um conjunto de características do código-fonte de um *software* desenvolvido dentro dos conceitos de orientação a objetos, verificou-se através de um experimento controlado que é possível avaliar a tendência da manutenibilidade do *software*. Para a avaliação desta tendência, uma análise estatística considerou as faixas de valores sugeridos para cada uma das métricas de C&K distribuídos ao longo dos trabalhos pesquisados.

À medida que o experimento foi sendo executado, observou-se que a obtenção dos valores de cada uma das métricas não é uma tarefa complexa principalmente quando é utilizada uma ferramenta específica e construída para apoiar esta tarefa, mas requer cuidado e organização em sua aplicação devido ao grande volume de dados manipulados. Há várias destas ferramentas disponíveis e a que foi escolhida para uso no experimento - CKJM - Chidamber & Kemerer Java Metrics - mostrou-se simples no uso, objetiva ao calcular os valores das métricas e confiável quanto ao valor fornecido, já que uma pequena amostra pôde ter seu valor comparado manualmente.

De todos os valores das métricas que foram calculados, uma parte é apresentada no formato de tabela junto aos Apêndices C, D e E. Esta é uma contribuição deste trabalho já que, nos trabalhos pesquisados se encontrou pouca informação deste tipo que pudesse ser reutilizada. Um dos trabalhos que trouxe os valores das métricas, e é referenciado em outros, é o de Li & Henry (WEI e SALLIE, 1993) que apresentam as métricas calculadas dos *softwares* UIMS – User Interface System e QUES – Quality Evaluation System.

Um ponto interessante observado no experimento é que o software utilizado como base para a análise estatística – jUnit – tem várias versões cujos

códigos-fonte estão disponíveis e permitiram observar a variação dos valores das métricas de versão em versão, facilitando a avaliação, particularmente depois que os dados numéricos foram transcritos para gráficos que evidenciaram momentos onde as métricas apontaram para prováveis alterações importantes na estrutura do código do *software* jUnit.

Embora o conjunto de seis métricas proposto por C&K envolva algumas das principais características dos objetos em um ambiente de desenvolvimento com base em Orientação a Objetos (acoplamento, coesão e herança), não considera diretamente outras características como polimorfismo e encapsulamento. Para ampliar a cobertura das características dos objetos, encontraram-se ao longo dos trabalhos pesquisados sugestões para outras métricas. Li & Henry (WEI e SALLIE, 1993) sugerem o uso da métrica LOC (*Lines Of Code*) para medir o tamanho do *software*, e Zhou & Leung (ZHOU e LEUNG, 2006) sugerem, por exemplo, NOM (*Number Of Methods*) e CHANGE (*Number Of Lines Changed in the Class*).

Assim, pode-se afirmar que há uma relação forte entre o valor da métrica coletada a partir do código-fonte e a manutenibilidade do *software*. E que, também, é possível utilizar as métricas de C&K para prever a tendência da manutenibilidade do *software*, na forma de uma melhorada ou piorada, em média, não só ao final de cada versão como também ao longo do seu desenvolvimento ou de um processo de manutenção intermediário do *software*. Em relação ao *software* jUnit, sua manutenibilidade, em média, tem melhorado ao longo das versões mais recentes sendo que, os primeiros sinais desta melhorada, observados através dos valores das métricas, apareceram a partir da versão 3.8.1 quando, há indicações de uma forte reorganização interna de seu código que passou de algumas dezenas de classes para, nas versões posteriores, algumas centenas de classes.

7.2 Trabalhos Futuros

Como trabalhos futuros, pode-se sugerir:

- Expandir a análise estatística e incorporar outros modelos de análise ou algoritmos;

- Reproduzir o experimento para um maior número de *softwares*, componentes ou módulos e avaliar eventuais limites práticos do experimento utilizado;
- Replicar o estudo para *softwares* desenvolvidos com outras linguagens de programação, mas ainda junto ao conceito de orientação a objetos;
- Multiplicar o experimento para *softwares* com arquiteturas conhecidamente distintas;
- Relacionar manutenibilidade com complexidade a partir da arquitetura de *software*, para apoiar as atividades da fase de projeto.

REFERÊNCIAS

ABRAN, A.; MOORE, J. W.; BOURQUE, P.; DUPUIS, R. **SWEBOK - Guide to the Software Engineering Body of Knowledge**. Los Alamitos: IEEE - Computer Society, 2004. 204 p.

ABREU, F. B. E.; GOULÃO, M.; ESTEVES, R. **Toward the Design Quality Evaluation of Object-Oriented Software Systems**. Austin: IEEE-Proceedings of the 5th International Conference on Software Quality, 1995. 12 p.

AGGARWAL, K. K.; SINGH, Y.; CHHABRA, J. K. **An Integrated Measure of Software Maintainability**. Seattle: [s.n.], v. 0-7803-7348-0/02, 2002. 7 p. 2002 Proceedings Annual Reliability and Maintainability Symposium.

ALAGAR, V. S.; LI, Q.; ORMANDJIEVA, O. S. **Assessment of Maintainability in Object-Oriented Software**. Santa Barbara: [s.n.], v. 1530-2067/01, 2001. 12 p. 39th Int'l Conf. and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS'01).

ANDA, B. **Assessing Software System Maintainability using Structural Measures and Expert Assessments**. Paris: [s.n.], 2007. 10 p. 23rd International Conference on Software Maintenance (ICSM 2007).

ANDERSSON, M.; VESTERGREN, P. **Object-Oriented Design Quality Metrics**. Uppsala: Uppsala University, v. ISSN 1100-1836, 2004. 67 p. Uppsala Master's Theses in Computer Science 276.

ANTONELLIS, P.; ANTONIOU, D.; KANELLOPOULOS, Y.; MAKRIS, C.; THEODORIDIS, C.; TJORTJIS, C.; TSIRAKIS, N. **A Data Mining Methodology for Evaluating Maintainability according to ISO/IEC-9126 Software Engineering-Product Quality Standard**. Amsterdam: [s.n.], 2007. 9 p. 11th European Conference on Software Maintenance and Reengineering (CSMR 2007).

BASILI, V. R.; BRIAND, L. C.; MELO, W. L. **A Validation of Object-Oriented Design Metrics as Quality Indicators**. [S.l.]: IEEE-Transactions On Software Engineering, v. 22-10, 1996. 11 p.

BORGES, B. G. **Ferramenta de apoio à coleta de métricas em software orientado a objetos**. Blumenau: Universidade Rgional de Blumenau, 2006. 68 p.

BRIAND, L.; DEVANBU, P.; MELO, W. **An Investigation Into Coupling Measures for C++**. New York: [s.n.], 1997. 10 p. Proceedings of the 19th International Conference on Software Engineering (ICSE '97).

BRIAND, L.; MORASCA, S.; BASILI, V. R. **Defining and Validating High-Level Design Metrics**. [S.l.]: ACM, v. CS-TR-3301, 1994. 32 p.

CHAHAL, K. K.; SINGH, H. **Metrics to Study Symptoms of bad Software Designs**. Amritsar: ACM, v. 34-1, 2009. 4 p.

CHIDAMBER, S. R.; DARCY, D. P.; KEMERER, C. F. **Managerial Use of Metrics for Object-Oriented Software - An Exploratory Analysis**. 0098-5589/98. ed. [S.l.]: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, v. 24-8, 1998. 11 p.

CHIDAMBER, S. R.; KEMERER, C. F. **A Metrics Suite for Object Oriented Design**. 6. ed. [S.l.]: [s.n.], v. 20, 1994. 18 p. IEEE Transactions on Software Engineering.

CHIDAMBER, S.; KEMERER, C. F. **Towards a Metrics Suite for Object Oriented Design**. Massachusetts: [s.n.], 1991. Sixth Annual ACM Conference on Object Oriented Programing, Systems, Languages and Applications.

COLEMAN, D.; ASH, D.; LOWTHER, B.; OMAN, P. **Using Metrics to Evaluate Software System Maintainability**. [S.l.]: [s.n.], v. 0018-9162/94, 1994. 6 p.

DARCY, D. P.; KEMERER, C. F. **OO Metrics in Practice**. November/December. ed. [S.l.]: IEEE, 2005. 3 p.

DASKALANTONAKIS, M. K. **A Pratical View of Software Measurement and Implementation Experiences Within Motorola**. Arlington Heights: [s.n.], v. VOL. 18, Nº 11, 1992. 13 p. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING.

DEMARCO, T. **Controlling Software Projects - Management, Measurement and Estimates**. Englewood Cliffs: [s.n.], 1986.

ELISH, M. O.; ELISH, K. O. **Application of TreeNet in Predicting Object-oriented Software Maintainability - A Comparative Study**. Fraunhofer IESE: [s.n.], 2009. 9 p. In Proceedings of the 13th IEEE European Conference on Software Maintenance and Reengineering (CSMR'09).

EMAM, K. E. **Object-Oriented Metrics - A review of Theory and Practice**. [S.l.]: Institute for Information Technology, v. NRC-44190, 2001. 32 p.

EMAM, K. E.; BENLARBI, S.; GOEL,N.; RAI, S. N. **The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics**. 0098-5589/01. ed. New York: [s.n.], v. 27-01, 2001.

FOWLER, M.; BECK, K.; BRANT, J.; OPDYKE, W.; ROBERTS, D. **Refactoring-Improving the Design of Existing Code**. Don Mills: [s.n.], 1999. 464 p.

GENERO, M.; PIATTINI, M.; CALERO, C. **A Survey of Metrics for UML Class Diagrams**. [S.l.]: [s.n.], 2005. 15 p.

GROGONO, P. **The Evolution of Programming Languages**. Montreal: [s.n.], 2002. 125 p.

HALSTEAD, M. H. **Elements of Software Science (Operating and programming systems series)**. New York: [s.n.], 1977. 142 p.

HARRISON, R.; COUNSELL, S.; NITHI, R. **An Overview of Object-Oriented Design metrics**. Southampton: IEEE, v. 0-8186-7840-2, 1997. 6 p.

HARRISON, R.; COUNSELL, S.; NITHI, R. **Coupling Metrics for Object-Oriented Design**. Bethesda: [s.n.], 1998. 8 p. Proceedings Fifth International Software Metrics Symposium Metrics 1998.

HEITLAGER, I.; KUIPERS, T.; VISSER, J. **A Practical Model for Measuring Maintainability**. Lisboa: [s.n.], 2007. 10 p. Sixth International Conference on the Quality of Information and Communications Technology.

HENDERSON-SELLERS, B. **Object-oriented metrics - measures of complexity**. Upper Saddle River: [s.n.], 1996. 252 p.

HONGLEI, T.; WEI, S.; YANAN, Z. **The Research on Software Metrics and Software Complexity Metrics**. Chongqing: [s.n.], 2009. 6 p. 2009 International Forum on Computer Science-Technology and Applications.

HUNG, M.; ZOU, Y. **Extracting Business Processes from Three-Tier Architecture Systems**. Kingston: Department of Electrical and Computer Engineering Queen’s University, 2005. 5 p.

IEEE STD 1061. New York: [s.n.], 2004. 26 p. **Standard for a Software Quality Metrics Methodology.**

IEEE STD 1219. Los Alamitos: [s.n.], 1993. 26 p. **Standard for Software Maintenance.**

IEEE STD 610. New York: [s.n.], 1990. 84 p. **Glossary of Software Engineering Teminology.**

INSTITUTO ANTÔNIO HOUAISS. **Dicionário Houaiss da Língua Portuguesa**. [S.l.]: Editora Objetiva, 2009.

ISO/IEC 9126-1. New York: [s.n.], 2001. 32 p. **Software Engineering - Product Quality - Part 1: Quality model.**

ISO/IEC 9126-2. New York: [s.n.], 2002. **Software Engineering - Product Quality - Part 2: External Metrics.**

JONES, C. **The Economics of Software Maintenance in the Twenty First Century**. Hendersonville: [s.n.], 2006. 19 p.

KANELLOPOULOS, Y.; DIMOPULOS, T.; TJORTJIS, C.; MAKRIS, C. **Mining Source Code Elements for Comprehending Object-Oriented Systems and Evaluating Their Maintainability**. New York: [s.n.], v. Volume 8, Issue 1, 2006. 8 p. ACM SIGKDD Explorations Newsletter.

KANELLOPOULOS, Y.; HEITLAGER, I. **Interpretation of Source Code Clusters in Terms of the ISO/IEC-9126 Maintainability Characteristics**. Athens: [s.n.], v. 978-1-4244-2157-2/08, 2008. 10 p.

KJELLQVIST, H. **Object Oriented Metrics - An Overview**. Västeras: [s.n.], 2008. 4 p.

KULKARNI, U. L.; KALSHETTY, Y. R.; ARDE, V. G. **Validation of CK metrics for Object Oriented Design Measurement**. Pilani: [s.n.], v. 978-0-7695-4246-1/10, 2010. 6 p. Third International Conference on Emerging Trends in Engineering and Technology.

LANZA, M.; GALL, H.; DUGERDIL, P. **EvoSpaces-Multi-dimensional Navigation Spaces for Software Evolution**. [S.l.]: IEEE Computer Society, 2009. 4 p. European Conference on Software Maintenance and Reengineerin.

LANZA, M.; MARINESCU, R. **Object-Oriented Metrics in Practice**. Berlin: Springer-Verlag, 2006. 205 p.

LINCKE, R.; GUTZMANN, T.; LÖWE, W. **Software Quality Prediction Models Compared**. Zhangjiajie: [s.n.], 2010. 10 p. Proceedings of the 2010 10th International Conference on Quality Software.

LINCKE, R.; LUNDBERG, J.; LÖWE, W. **Comparing Software Metrics Tools**. Seattle: [s.n.], v. 978-1-59593-904-3/08/07, 2008. 11 p. Proceedings of the 2008 workshop on Testing, analysis, and verification of web services and applications (ISSTA'08).

LORENZ, M.; KIDD, J. **Object-oriented Software Metrics**. Englewood Cliffs: [s.n.], 1994. 2 p.

MADEYSKI, L. **The Impact of Pair Programming and Test-Driven Development on Package Dependencies in Object-Oriented Design - An Experiment**. Wroclaw: Institute of Applied Informatics, 2006. 12 p.

MCCABE, T. J. **A Complexity Measure**. Ft. Meade: [s.n.], v. SE-2 - Nº 4, 1976. 13 p.

MCCONNELL, S. **Code Complete - A Practical Handbook of Software Construction**. [S.l.]: Microsoft Press, 2004. 960 p. ISBN-13: 978-0735619678.

MUTHANNA, S.; KONTOGIANNIS, K.; PONNAMBALAM, K.; STACEY, B. **A Maintainability Model for Industrial Software Systems Using Design Level Metrics**. Brisbane: [s.n.], v. 1095-1350/00, 2000. 9 p. Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00).

ORDONEZ, M. J.; HADDAD, H. M. **The State of Metrics in Software Industry**. Las Vegas: [s.n.], 2008. 6 p. Fifth International Conference on Information Technology: New Generations.

PEARSON, K. **The Grammar of Science**. [S.l.]: [s.n.], 1892.

PIGOSKI, T. M. **Practical Software Maintenance**. [S.l.]: Wiley Computer Publishing, 1997. 400 p.

PRESSMAN, R. S. **A Practitioner's Approach**. 5ª. ed. New York: McGraw-Hill Higher Education, 2001. 888 p.

REDDY, K. N.; RAO, A. A. **Dependecy Oriented Complexity Metrics to Detect Rippling Related Design Defects**. July. ed. [S.l.]: SIGSOFT Software Engineering Notes, v. 34-4, 2009. 7 p.

SATC-SOFTWARE ASSURANCE TECHNOLOGY CENTER. **Software Quality Metrics for Object Oriented System Environments**. Greenbelt: NASA-National Aeronautics and Space Administration, 1995. 24 p. SATC-TR-95-1001.

SELVARANI, R.; NAIR, T. R. G.; PRASAD, V. K. **Estimation of Defect proneness Using Design complexity Measurements in Object-Oriented Software**. [S.l.]: [s.n.], v. 978-0-7695-3654-5/09, 2009. 5 p. International Conference on Signal Processing Systems.

SINGH, P.; SINGH, H. **DynaMetrics - A Runtime Metric-Based Analysis Tool for Object-Oriented Software Systems**. New York: ACM-Software Engineering Notes, v. 33-6, 2008. 6 p.

STAPEL, K.; LÜBKE, D.; KNAUSS, E. **Best Practices in eXtreme Programming Course Design**. Leipzig: [s.n.], v. 978-1-60558-079-1/08/05, 2008. 30th International Conference on Software Engineering ® - (ICSE'08).

SUBRAMANYAM, R.; KRISHNAN, M. S. **Empirical Analysis of CK Metrics for Object-Oriented Design Complexity:** Implications for Software Defects. 0098-5589/03. ed. [S.l.]: IEEE-TRANSACTIONS ON SOFTWARE ENGINEERING, v. 29-4, 2003. 14 p.

SWANSON, E. B. **The Dimensions of Maintenance**. Los Angeles: [s.n.], 1976. Proceeding ICSE '76 Proceedings of the 2nd International Conference on Software Engineering.

TANG, M.-H.; KAO, M.-H.; CHEN, M.-H. **An Empirical Study on Object-Oriented Metrics**. Boca Raton: [s.n.], 1999. 8 p. Proceedings Sixth International Software Metrics Symposium.

VAN KOTEN, C.; GRAY, A. **An Application of Bayesian Network for Predicting Object-Oriented Software Maintainability**. Dunedin: University of Otago, 2005. 21 p.

VERHOEF, C. **Software Evolution - A Taxonomy**. Amsterdam: University of Amsterdam Programming Research Group, 2000. 5 p.

WANG, L.-J.; HU, X.-X.; NING, Z.-Y.; KE, W.-H. **Predicting Object-Oriented Software Maintainability using Projection Pursuit Regression**. 978-0-7695-3887-7/09. ed. Nanjing: [s.n.], 2009. 4 p.

WEI, L.; SALLIE, H. **Object-Oriented Metrics that Predict Maintainability**. New York: [s.n.], 1993. 12 p.

YU, S.; ZHOU, S. **A Survey on Metric of Software Complexity**. Chengdu: [s.n.], v. 978-1-4244-5265-1/10, 2010. 5 p.

ZHOU, Y.; LEUNG, H. **Predicting object-oriented software maintainability using multivariate adaptive regression splines**. New York: [s.n.], v. Vol. 80 Issue 8, 2006. 13 p. Journal of Systems and Software.

ZHUO, F.; LOWTHER, B.; OMAN, P.; HAGEMEISTER, J. **Constructing and Testing Software Maintainability Assessment Models**. Baltimore: [s.n.], 1993. 10 p. Proceedings First International Software Metrics Symposium, 1993.

ZUSE, H. **A Framework of Software Measurement**. Berlin: Walter de Gruyter, 1998. 755 p.

ZVEGINTZOV, N.; PARIKH, G. **60 years of Software Maintenance-Lessons Learned**. [S.l.]: [s.n.], 2005. 2 p.

ANEXO A – Tabela com os valores da Correlação de Pearson

Tabela 15: Valores da Correlação de Pearson calculada para as 21 versões do *software* jUnit.

Versão do *software* jUnit	Correlações de Pearson									
	WMC-DIT	WMC-CBO	WMC-RFC	WMC-LCOM	DIT-CBO	DIT-RFC	DIT-LCOM	CBO-RFC	CBO-LCOM	RFC-LCOM
2.0	-0,04	0,57	0,94	0,61	0,11	0,02	0,00	0,69	0,72	0,61
2.1	-0,05	0,70	0,96	0,68	0,07	-0,01	0,04	0,76	0,86	0,66
3.0	-0,06	0,64	0,95	0,66	0,12	-0,03	0,02	0,74	0,74	0,62
3.2	-0,06	0,64	0,95	0,66	0,12	-0,03	0,02	0,74	0,74	0,62
3.4	-0,06	0,47	0,95	0,68	0,11	-0,04	-0,01	0,60	0,47	0,62
3.5	-0,05	0,44	0,94	0,68	0,10	-0,02	-0,01	0,59	0,42	0,61
3.6	-0,04	0,42	0,94	0,68	0,10	-0,02	0,01	0,58	0,33	0,57
3.7	-0,04	0,42	0,94	0,68	0,10	-0,02	0,01	0,58	0,33	0,57
3.8	0,05	0,42	0,90	0,76	0,07	0,06	0,05	0,66	0,18	0,54
3.8.1	0,04	0,49	0,92	0,79	0,07	0,05	0,03	0,69	0,45	0,71
3.8.2	0,04	0,46	0,90	0,81	0,06	0,06	0,03	0,66	0,40	0,64
4.0	0,07	0,34	0,88	0,81	0,20	0,09	0,01	0,63	0,12	0,60
4.1	0,08	0,34	0,88	0,81	0,21	0,11	0,02	0,64	0,12	0,60
4.2	0,08	0,35	0,89	0,80	0,22	0,11	0,01	0,64	0,13	0,60
4.3.1	0,09	0,30	0,89	0,80	0,23	0,11	0,00	0,59	0,06	0,62
4.4	0,08	0,38	0,90	0,79	0,20	0,10	0,00	0,65	0,13	0,63
4.5	0,06	0,39	0,89	0,78	0,18	0,09	0,00	0,67	0,13	0,60
4.6	0,05	0,40	0,88	0,77	0,16	0,07	-0,01	0,69	0,11	0,58
4.7	0,06	0,42	0,89	0,77	0,15	0,08	0,00	0,69	0,12	0,58
4.8	0,07	0,44	0,89	0,76	0,16	0,09	0,00	0,70	0,12	0,57
4.9b2	0,07	0,44	0,89	0,76	0,16	0,09	0,00	0,69	0,13	0,58

ANEXO B – Métricas obtidas usando a ferramenta CKJM e a versão 2.0 do *software* jUnit

Tabela 16: Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 31 classes da versão 2.0 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
1	junit.samples.AllTests	3	1	0	6	10	3
2	junit.samples.SimpleTest	6	2	0	3	17	13
3	junit.samples.VectorTest	10	2	0	4	29	3
4	junit.samples.money.IMoney	7	1	0	2	7	21
5	junit.samples.money.Money	13	1	0	2	26	68
6	junit.samples.money.MoneyBag	19	1	0	2	41	0
7	junit.samples.money.MoneyTest	22	2	0	7	44	33
8	junit.tests.AllTests	3	1	0	5	9	3
9	junit.tests.ClassLoaderTestCase	4	1	0	0	12	6
10	junit.tests.NoTestCaseClass	2	1	0	0	3	1
11	junit.tests.NoTestCases	2	2	0	1	3	1
12	junit.tests.NotPublicTestCase	3	2	0	1	4	3
13	junit.tests.NotVoidTestCase	3	2	0	1	4	3
14	junit.tests.OneTestCase	4	2	0	1	6	6
15	junit.tests.SuiteTest	9	2	0	4	24	16
16	junit.tests.TestTest$1	2	2	0	1	5	1
17	junit.tests.TestTest$2	2	2	0	1	4	1
18	junit.tests.TestTest$3	2	2	0	1	4	1
19	junit.tests.TestTest$4	2	2	0	1	4	1
20	junit.tests.TestTest$5	3	0	0	1	6	3
21	junit.tests.TestTest$6	3	2	0	1	5	3
22	junit.tests.TestTest$7	2	2	0	1	4	1
23	junit.tests.TestTest$8	3	2	0	1	5	3
24	junit.tests.TestTest$9	2	0	0	1	4	1
25	junit.tests.TestTest$TornDown	3	2	0	1	5	1
26	junit.tests.TestTest	15	2	0	14	39	105
27	junit.tests.TestTestCaseClassLoader	2	2	0	2	15	1
28	junit.tests.ThreadTest$1	2	2	0	1	4	0
29	junit.tests.ThreadTest$2	2	2	0	1	4	0
30	junit.tests.ThreadTest$3	2	2	0	1	4	0
31	junit.tests.ThreadTest	5	2	0	7	21	10

ANEXO C – Métricas obtidas usando a ferramenta CKJM e a versão 3.8 do *software* jUnit

Tabela 17: Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 79 classes da versão 3.8 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
1	junit.samples.AllTests	3	1	0	6	13	3
2	junit.samples.SimpleTest	7	3	0	6	20	19
3	junit.samples.VectorTest	10	3	0	6	29	3
4	junit.samples.money.IMoney	8	1	0	2	8	28
5	junit.samples.money.Money	14	1	0	2	28	81
6	junit.samples.money.MoneyBag	18	1	0	2	42	43
7	junit.samples.money.MoneyTest	25	3	0	6	51	26
8	junit.tests.AllTests	3	1	0	7	10	3
9	junit.tests.WasRun	2	3	0	1	3	0
10	junit.tests.extensions.ActiveTestTest$SuccessTest	2	3	0	1	3	1
11	junit.tests.extensions.ActiveTestTest	6	3	0	8	17	15
12	junit.tests.extensions.AllTests	3	1	0	4	10	3
13	junit.tests.extensions.ExceptionTestCaseTest$ThrowExceptionTestCase	2	4	0	1	4	1
14	junit.tests.extensions.ExceptionTestCaseTest$ThrowNoExceptionTestCase	2	4	0	1	3	1
15	junit.tests.extensions.ExceptionTestCaseTest$ThrowRuntimeExceptionTestCase	2	4	0	1	4	1
16	junit.tests.extensions.ExceptionTestCaseTest	6	3	0	6	20	3
17	junit.tests.extensions.ExtensionTest$1	2	3	0	2	4	1
18	junit.tests.extensions.ExtensionTest$2	2	3	0	2	4	1
19	junit.tests.extensions.ExtensionTest$3	2	3	0	1	4	1
20	junit.tests.extensions.ExtensionTest$4	2	0	0	3	4	1
21	junit.tests.extensions.ExtensionTest$5	2	4	0	3	4	1
22	junit.tests.extensions.ExtensionTest$TornDown	2	4	0	2	3	0
23	junit.tests.extensions.ExtensionTest	5	3	0	13	22	10
24	junit.tests.extensions.RepeatedTestTest$SuccessTest	2	3	0	1	3	1
25	junit.tests.extensions.RepeatedTestTest	5	3	0	7	16	0

Tabela 17 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 79 classes da versão 3.8 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
26	junit.tests.framework.AllTests	4	1	0	5	15	6
27	junit.tests.framework.AssertTest	33	3	0	4	48	528
28	junit.tests.framework.Failure	2	3	0	2	4	1
29	junit.tests.framework.InheritedTestCase	2	0	0	1	3	1
30	junit.tests.framework.NoArgTestCaseTest	2	3	0	1	3	1
31	junit.tests.framework.NoTestCaseClass	2	1	0	0	3	1
32	junit.tests.framework.NoTestCases	2	3	0	1	3	1
33	junit.tests.framework.NotPublicTestCase	3	3	0	1	4	3
34	junit.tests.framework.NotVoidTestCase	3	3	0	1	4	3
35	junit.tests.framework.OneTestCase	4	3	0	1	5	6
36	junit.tests.framework.OverrideTestCase	2	0	0	1	3	1
37	junit.tests.framework.Success	3	3	0	1	4	3
38	junit.tests.framework.SuiteTest	12	3	0	5	31	10
39	junit.tests.framework.TestCaseTest$1	2	3	0	1	4	1
40	junit.tests.framework.TestCaseTest$2	3	0	0	1	6	3
41	junit.tests.framework.TestCaseTest$3	3	3	0	1	5	3
42	junit.tests.framework.TestCaseTest$4	2	3	0	1	3	1
43	junit.tests.framework.TestCaseTest$5	2	3	0	2	4	1
44	junit.tests.framework.TestCaseTest$6	3	3	0	1	5	3
45	junit.tests.framework.TestCaseTest$7	2	0	0	1	4	1
46	junit.tests.framework.TestCaseTest$8	2	0	0	1	4	1
47	junit.tests.framework.TestCaseTest$9	1	3	0	1	2	0
48	junit.tests.framework.TestCaseTest$Torn Down	3	3	0	1	5	1
49	junit.tests.framework.TestCaseTest	17	3	0	18	46	136
50	junit.tests.framework.TestImplementorTest $1	2	1	0	3	5	0
51	junit.tests.framework.TestImplementorTest $2	2	3	0	1	3	1
52	junit.tests.framework.TestImplementorTest $DoubleTestCase	4	1	0	5	9	4
53	junit.tests.framework.TestImplementorTest	2	3	0	5	12	0
54	junit.tests.framework.TestListenerTest$1	2	3	0	1	4	1

Tabela 17 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 79 classes da versão 3.8 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
55	junit.tests.framework.TestListenerTest$2	2	3	0	2	4	1
56	junit.tests.framework.TestListenerTest$3	2	3	0	1	3	1
57	junit.tests.framework.TestListenerTest	9	3	0	9	17	6
58	junit.tests.runner.AllTests	4	1	0	5	16	6
59	junit.tests.runner.BaseTestRunnerTest$MockRunner	5	2	0	3	6	10
60	junit.tests.runner.BaseTestRunnerTest$NonStatic	2	1	0	1	3	1
61	junit.tests.runner.BaseTestRunnerTest	4	3	0	4	12	4
62	junit.tests.runner.ClassLoaderTest	5	2	0	1	14	10
63	junit.tests.runner.LoadedFromJar	4	2	0	1	13	6
64	junit.tests.runner.SimpleTestCollectorTest	2	3	0	4	8	1
65	junit.tests.runner.SorterTest$Swapper	2	1	0	1	5	1
66	junit.tests.runner.SorterTest	2	3	0	5	10	1
67	junit.tests.runner.StackFilterTest	3	3	0	3	10	1
68	junit.tests.runner.TestCaseClassLoaderTest	4	3	0	3	19	6
69	junit.tests.runner.TextFeedbackTest$1	2	3	0	1	3	1
70	junit.tests.runner.TextFeedbackTest$2	2	3	0	1	3	1
71	junit.tests.runner.TextFeedbackTest$3	2	3	0	1	3	1
72	junit.tests.runner.TextFeedbackTest$4	2	0	0	4	5	1
73	junit.tests.runner.TextFeedbackTest$5	2	3	0	2	4	1
74	junit.tests.runner.TextFeedbackTest$6	2	0	0	4	5	1
75	junit.tests.runner.TextFeedbackTest$7	2	3	0	1	4	1
76	junit.tests.runner.TextFeedbackTest$TestResultPrinter	2	2	0	2	3	1
77	junit.tests.runner.TextFeedbackTest	9	3	0	15	33	6
78	junit.tests.runner.TextRunnerTest$1	2	2	0	0	3	1
79	junit.tests.runner.TextRunnerTest	6	3	0	7	26	15

ANEXO D – Métricas obtidas usando a ferramenta CKJM e a versão 4.9b2 do *software* jUnit

Tabela 18: Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
1	junit.samples.AllTests	3	1	0	6	10	3
2	junit.samples.ListTest	9	3	0	5	25	6
3	junit.samples.package-info	0	1	0	0	0	0
4	junit.samples.SimpleTest	7	3	0	5	17	19
5	junit.samples.money.IMoney	8	1	0	2	8	28
6	junit.samples.money.Money	14	1	0	2	30	81
7	junit.samples.money.MoneyBag	18	1	0	2	42	43
8	junit.samples.money.MoneyTest	25	3	0	5	48	26
9	junit.samples.money.package-info	0	1	0	0	0	0
10	junit.tests.AllTests	3	1	0	7	10	3
11	junit.tests.package-info	0	1	0	0	0	0
12	junit.tests.WasRun	2	3	0	1	3	0
13	junit.tests.extensions.ActiveTestTest$SuccessTest	2	3	0	1	3	1
14	junit.tests.extensions.ActiveTestTest	6	3	0	6	17	15
15	junit.tests.extensions.AllTests	3	1	0	4	7	3
16	junit.tests.extensions.ExtensionTest$1	2	3	0	2	4	1
17	junit.tests.extensions.ExtensionTest$2	2	3	0	2	4	1
18	junit.tests.extensions.ExtensionTest$3	2	3	0	2	4	1
19	junit.tests.extensions.ExtensionTest$4	2	0	0	3	4	1
20	junit.tests.extensions.ExtensionTest$5	2	4	0	3	4	1
21	junit.tests.extensions.ExtensionTest$TornDown	2	4	0	2	3	0
22	junit.tests.extensions.ExtensionTest	5	3	0	12	23	10
23	junit.tests.extensions.package-info	0	1	0	0	0	0
24	junit.tests.extensions.RepeatedTestTest$SuccessTest	2	3	0	1	3	1
25	junit.tests.extensions.RepeatedTestTest	5	3	0	6	19	0
26	junit.tests.framework.AllTests	3	1	0	5	9	3
27	junit.tests.framework.AssertTest	17	3	0	3	34	136
28	junit.tests.framework.ComparisonCompactorTest	20	3	0	2	26	190

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
29	junit.tests.framework.ComparisonFailureTest	5	3	0	2	18	10
30	junit.tests.framework.DoublePrecisionAssertTest	8	3	0	2	11	28
31	junit.tests.framework.Failure	2	3	0	1	4	1
32	junit.tests.framework.FloatAssertTest	9	3	0	2	13	36
33	junit.tests.framework.InheritedTestCase	2	0	0	1	3	1
34	junit.tests.framework.NoArgTestCaseTest	2	3	0	1	3	1
35	junit.tests.framework.NoTestCaseClass	2	1	0	0	3	1
36	junit.tests.framework.NoTestCases	2	3	0	1	3	1
37	junit.tests.framework.NotPublicTestCase	3	3	0	1	4	3
38	junit.tests.framework.NotVoidTestCase	3	3	0	1	4	3
39	junit.tests.framework.OneTestCase	4	3	0	1	5	6
40	junit.tests.framework.OverrideTestCase	2	0	0	1	3	1
41	junit.tests.framework.package-info	0	1	0	0	0	0
42	junit.tests.framework.Success	3	3	0	1	4	3
43	junit.tests.framework.SuiteTest	14	3	0	4	38	35
44	junit.tests.framework.TestCaseTest$1	2	3	0	2	4	1
45	junit.tests.framework.TestCaseTest$10	1	3	0	2	2	0
46	junit.tests.framework.TestCaseTest$2	3	0	0	2	6	3
47	junit.tests.framework.TestCaseTest$3	3	3	0	2	5	3
48	junit.tests.framework.TestCaseTest$4	2	3	0	2	3	1
49	junit.tests.framework.TestCaseTest$5	2	3	0	2	4	1
50	junit.tests.framework.TestCaseTest$6	3	3	0	2	5	3
51	junit.tests.framework.TestCaseTest$7	2	0	0	2	4	1
52	junit.tests.framework.TestCaseTest$8	2	0	0	2	4	1
53	junit.tests.framework.TestCaseTest$9	3	3	0	2	5	1
54	junit.tests.framework.TestCaseTest$TornDown	3	3	0	1	5	1
55	junit.tests.framework.TestCaseTest	18	3	0	17	50	153
56	junit.tests.framework.TestImplementorTest$1	2	3	0	2	3	1
57	junit.tests.framework.TestImplementorTest$DoubleTestCase$1	2	1	0	3	5	0

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
58	junit.tests.framework.TestImplementorTest$DoubleTestCase	4	1	0	5	9	4
59	junit.tests.framework.TestImplementorTest	2	3	0	4	12	0
60	junit.tests.framework.TestListenerTest$1	2	3	0	2	4	1
61	junit.tests.framework.TestListenerTest$2	2	3	0	2	4	1
62	junit.tests.framework.TestListenerTest$3	2	3	0	2	3	1
63	junit.tests.framework.TestListenerTest	9	3	0	8	17	6
64	junit.tests.framework.ThreeTestCases	4	3	0	1	5	6
65	junit.tests.runner.AllTests	4	1	0	4	10	6
66	junit.tests.runner.BaseTestRunnerTest$DoesntExtendTestCase	2	1	0	2	4	1
67	junit.tests.runner.BaseTestRunnerTest$MockRunner	6	2	0	2	7	9
68	junit.tests.runner.BaseTestRunnerTest$NonStatic	2	1	0	1	3	1
69	junit.tests.runner.BaseTestRunnerTest	3	3	0	4	10	3
70	junit.tests.runner.package-info	0	1	0	0	0	0
71	junit.tests.runner.StackFilterTest	3	3	0	2	10	1
72	junit.tests.runner.TextFeedbackTest$1	2	3	0	2	3	1
73	junit.tests.runner.TextFeedbackTest$2	2	3	0	2	3	1
74	junit.tests.runner.TextFeedbackTest$3	2	3	0	2	3	1
75	junit.tests.runner.TextFeedbackTest$4	2	0	0	3	5	1
76	junit.tests.runner.TextFeedbackTest$5	2	3	0	3	4	1
77	junit.tests.runner.TextFeedbackTest$6	2	0	0	3	5	1
78	junit.tests.runner.TextFeedbackTest$7	2	3	0	2	4	1
79	junit.tests.runner.TextFeedbackTest$TestResultPrinter	2	2	0	1	3	1
80	junit.tests.runner.TextFeedbackTest	9	3	0	14	29	6

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
81	junit.tests.runner.TextRunnerSingleMethodTest$InvocationTest	3	3	0	2	5	3
82	junit.tests.runner.TextRunnerSingleMethodTest	2	3	0	4	10	1
83	junit.tests.runner.TextRunnerTest$1	2	2	0	1	3	1
84	junit.tests.runner.TextRunnerTest	6	3	0	6	25	15
85	org.junit.samples.ListTest	11	1	0	5	28	13
86	org.junit.samples.package-info	0	1	0	0	0	0
87	org.junit.samples.SimpleTest	5	1	0	3	12	10
88	org.junit.samples.money.MoneyTest	26	1	0	6	48	51
89	org.junit.samples.money.package-info	0	1	0	0	0	0
90	org.junit.tests.AllTests	2	1	0	2	4	1
91	org.junit.tests.ObjectContractTest	5	1	0	5	15	10
92	org.junit.tests.package-info	0	1	0	0	0	0
93	org.junit.tests.ParentRunnerTest$1	3	1	0	2	8	3
94	org.junit.tests.ParentRunnerTest$FruitTest	3	1	0	1	7	3
95	org.junit.tests.ParentRunnerTest	3	1	0	7	10	1
96	org.junit.tests.TestSystem	4	1	0	1	7	2
97	org.junit.tests.assertion.AssertionTest	66	1	0	4	106	2145
98	org.junit.tests.assertion.BothTest	9	1	0	6	23	36
99	org.junit.tests.assertion.EachTest	2	1	0	4	7	1
100	org.junit.tests.assertion.MultipleFailureExceptionTest$ExpectedException	1	4	0	0	2	0
101	org.junit.tests.assertion.MultipleFailureExceptionTest	4	1	0	5	20	6
102	org.junit.tests.deprecated.JUnit4ClassRunnerTest$Example	3	1	0	1	5	3
103	org.junit.tests.deprecated.JUnit4ClassRunnerTest$UnconstructableExample	3	1	0	1	6	3
104	org.junit.tests.deprecated.JUnit4ClassRunnerTest	3	1	0	5	10	3
105	org.junit.tests.description.AnnotatedDescriptionTest$1	3	1	0	2	4	3
106	org.junit.tests.description.AnnotatedDescriptionTest$AnnotatedClass	2	1	0	0	3	1
107	org.junit.tests.description.AnnotatedDescriptionTest$IgnoredClass	2	1	0	0	3	1

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
108	org.junit.tests.description.AnnotatedDescriptionTest$MyOwnAnnotation	0	1	0	0	0	0
109	org.junit.tests.description.AnnotatedDescriptionTest$ValueAnnotatedClass	2	1	0	0	3	1
110	org.junit.tests.description.AnnotatedDescriptionTest$ValuedAnnotation	1	1	0	0	1	0
111	org.junit.tests.description.AnnotatedDescriptionTest	8	1	0	6	22	28
112	org.junit.tests.description.SuiteDescriptionTest	4	1	0	2	15	0
113	org.junit.tests.description.TestDescriptionTest	2	1	0	2	8	1
114	org.junit.tests.experimental.AssumptionTest$1	2	2	0	3	4	1
115	org.junit.tests.experimental.AssumptionTest$AssumptionFailureInConstructor	2	1	0	2	5	1
116	org.junit.tests.experimental.AssumptionTest$HasFailingAssumeInBefore	3	1	0	2	6	3
117	org.junit.tests.experimental.AssumptionTest$HasFailingAssumeInBeforeClass	3	1	0	2	6	3
118	org.junit.tests.experimental.AssumptionTest$HasFailingAssumption	2	1	0	4	7	1
119	org.junit.tests.experimental.AssumptionTest$HasPassingAssumption	2	1	0	4	7	1
120	org.junit.tests.experimental.AssumptionTest	19	1	0	12	42	165
121	org.junit.tests.experimental.AssumptionViolatedExceptionTest	6	1	0	8	19	15
122	org.junit.tests.experimental.ExperimentalTests	1	1	0	0	2	0
123	org.junit.tests.experimental.MatcherTest	4	1	0	8	16	6
124	org.junit.tests.experimental.categories.CategoriesAndParameterizedTest$ParameterizedTestWithAttemptedMethodCategory	3	1	0	1	6	3
125	org.junit.tests.experimental.categories.CategoriesAndParameterizedTest$ParameterTokenSuiteMalformed	1	1	0	0	2	0

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
126	org.junit.tests.experimental.categories.CategoriesAndParameterizedTest$ParameterTokenSuiteMalformedAndSwapped	1	1	0	0	2	0
127	org.junit.tests.experimental.categories.CategoriesAndParameterizedTest$ParameterTokenSuiteWellFormed	1	1	0	0	2	0
128	org.junit.tests.experimental.categories.CategoriesAndParameterizedTest$Token	1	1	0	0	2	0
129	org.junit.tests.experimental.categories.CategoriesAndParameterizedTest$VanillaCategorizedJUnitTest	2	1	0	1	4	1
130	org.junit.tests.experimental.categories.CategoriesAndParameterizedTest$WellBehavedParameterizedTest	3	1	0	1	6	3
131	org.junit.tests.experimental.categories.CategoriesAndParameterizedTest	4	1	0	6	1	2
132	org.junit.tests.experimental.categories.CategoryTest$A	3	1	0	1	5	3
133	org.junit.tests.experimental.categories.CategoryTest$B	2	1	0	0	3	1
134	org.junit.tests.experimental.categories.CategoryTest$C	2	1	0	1	4	1
135	org.junit.tests.experimental.categories.CategoryTest$Category1	1	1	0	0	2	0
136	org.junit.tests.experimental.categories.CategoryTest$Category2	1	1	0	0	2	0
137	org.junit.tests.experimental.categories.CategoryTest$ChooseSlowFromBoth	1	1	0	0	2	0
138	org.junit.tests.experimental.categories.CategoryTest$ClassAsCategory	1	1	0	0	2	0
139	org.junit.tests.experimental.categories.CategoryTest$FastTests	0	1	0	0	0	0
140	org.junit.tests.experimental.categories.CategoryTest$IncludeAndExcludeSuite	1	1	0	0	2	0
141	org.junit.tests.experimental.categories.CategoryTest$JustA	1	1	0	0	2	0
142	org.junit.tests.experimental.categories.CategoryTest$OneFast	2	1	0	0	3	1

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
143	org.junit.tests.experimental.categories.CategoryTest$OneFastOneSlow	3	1	0	0	4	3
144	org.junit.tests.experimental.categories.CategoryTest$OneFastSuite	1	1	0	0	2	0
145	org.junit.tests.experimental.categories.CategoryTest$OneMoreTest	2	1	0	0	3	1
146	org.junit.tests.experimental.categories.CategoryTest$OneThatIsBothFastAndSlow	2	1	0	0	3	1
147	org.junit.tests.experimental.categories.CategoryTest$OneVerySlowTest	2	1	0	0	3	1
148	org.junit.tests.experimental.categories.CategoryTest$RunClassAsCategory	1	1	0	0	2	0
149	org.junit.tests.experimental.categories.CategoryTest$RunSlowFromVerySlow	1	1	0	0	2	0
150	org.junit.tests.experimental.categories.CategoryTest$SlowTests	0	1	0	0	0	0
151	org.junit.tests.experimental.categories.CategoryTest$SlowTestSuite	1	1	0	0	2	0
152	org.junit.tests.experimental.categories.CategoryTest$SomeAreSlow	6	1	0	0	7	15
153	org.junit.tests.experimental.categories.CategoryTest$SomeAreSlowSuite	1	1	0	0	2	0
154	org.junit.tests.experimental.categories.CategoryTest$TestSuiteWithNoCategories	1	1	0	0	2	0
155	org.junit.tests.experimental.categories.CategoryTest$VerySlowTests	0	1	0	1	0	0
156	org.junit.tests.experimental.categories.CategoryTest	13	1	0	12	37	78
157	org.junit.tests.experimental.max.DescriptionTest	4	1	0	2	9	6
158	org.junit.tests.experimental.max.JUnit38SortingTest$JUnit38Test	4	3	0	1	6	6
159	org.junit.tests.experimental.max.JUnit38SortingTest$JUnit4Test	2	1	0	0	3	1
160	org.junit.tests.experimental.max.JUnit38SortingTest	4	1	0	5	15	2
161	org.junit.tests.experimental.max.MaxStarterTest$1	2	2	0	3	5	0

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
162	org.junit.tests.experimental.max.MaxStarterTest$2	3	2	0	3	4	3
163	org.junit.tests.experimental.max.MaxStarterTest$HalfMalformedJUnit38TestMethod	3	3	0	1	4	3
164	org.junit.tests.experimental.max.MaxStarterTest$MalformedJUnit38Test	2	1	0	0	3	1
165	org.junit.tests.experimental.max.MaxStarterTest$MalformedJUnit38TestMethod	2	3	0	1	3	1
166	org.junit.tests.experimental.max.MaxStarterTest$TwoOldTests	3	3	0	1	4	3
167	org.junit.tests.experimental.max.MaxStarterTest$TwoTests	3	1	0	1	5	3
168	org.junit.tests.experimental.max.MaxStarterTest$TwoUnEqualTests	3	1	0	1	6	3
169	org.junit.tests.experimental.max.MaxStarterTest	20	1	0	17	70	4
170	org.junit.tests.experimental.parallel.ParallelClassTest$1	4	3	0	3	11	0
171	org.junit.tests.experimental.parallel.ParallelClassTest$Example1	2	1	0	0	4	1
172	org.junit.tests.experimental.parallel.ParallelClassTest$Example2	2	1	0	0	4	1
173	org.junit.tests.experimental.parallel.ParallelClassTest$ExampleSuite	1	1	0	0	2	0
174	org.junit.tests.experimental.parallel.ParallelClassTest	3	1	0	7	12	3
175	org.junit.tests.experimental.parallel.ParallelMethodTest$1	4	3	0	3	11	0
176	org.junit.tests.experimental.parallel.ParallelMethodTest$Example	3	1	0	0	5	3
177	org.junit.tests.experimental.parallel.ParallelMethodTest	3	1	0	7	12	3
178	org.junit.tests.experimental.results.PrintableResultTest$1	2	2	0	1	4	0
179	org.junit.tests.experimental.results.PrintableResultTest	4	1	0	8	15	6
180	org.junit.tests.experimental.results.ResultMatchersTest	3	1	0	5	13	3
181	org.junit.tests.experimental.rules.ClassRulesTest$Counter	2	3	0	1	3	0

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
182	org.junit.tests.experimental.rules.ClassRulesTest$CustomCounter$1	2	2	0	2	4	0
183	org.junit.tests.experimental.rules.ClassRulesTest$CustomCounter	2	2	0	4	4	1
184	org.junit.tests.experimental.rules.ClassRulesTest$ExampleTestWithClassRule	4	1	0	2	7	0
185	org.junit.tests.experimental.rules.ClassRulesTest$ExampleTestWithCustomClassRule	4	1	0	2	7	0
186	org.junit.tests.experimental.rules.ClassRulesTest$SubclassOfTestWithClassRule	1	0	0	1	2	0
187	org.junit.tests.experimental.rules.ClassRulesTest	4	1	0	7	9	6
188	org.junit.tests.experimental.rules.ExpectedExceptionRuleTest$1	4	3	0	2	7	0
189	org.junit.tests.experimental.rules.ExpectedExceptionRuleTest$ExpectedMessageMatcherFails	2	1	0	3	7	0
190	org.junit.tests.experimental.rules.ExpectedExceptionRuleTest$ExpectsMatcher	2	1	0	3	7	0
191	org.junit.tests.experimental.rules.ExpectedExceptionRuleTest$ExpectsMessageMatcher	2	1	0	3	7	0
192	org.junit.tests.experimental.rules.ExpectedExceptionRuleTest$ExpectsMultipleMatchers	2	1	0	1	7	0
193	org.junit.tests.experimental.rules.ExpectedExceptionRuleTest$ExpectsSubstring	2	1	0	1	6	0
194	org.junit.tests.experimental.rules.ExpectedExceptionRuleTest$ExpectsSubstringNullMessage	2	1	0	1	6	0
195	org.junit.tests.experimental.rules.ExpectedExceptionRuleTest$HasExpectedException	4	1	0	1	10	0
196	org.junit.tests.experimental.rules.ExpectedExceptionRuleTest$HasWrongExpectedException	2	1	0	1	6	0

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
197	org.junit.tests.experimental.rules.ExpectedExceptionRuleTest$HasWrongMessage	2	1	0	1	6	0
198	org.junit.tests.experimental.rules.ExpectedExceptionRuleTest$WronglyExpectsException	2	1	0	1	5	0
199	org.junit.tests.experimental.rules.ExpectedExceptionRuleTest$WronglyExpectsExceptionMessage	2	1	0	1	5	0
200	org.junit.tests.experimental.rules.ExpectedExceptionRuleTest	14	1	0	7	22	91
201	org.junit.tests.experimental.rules.ExternalResourceRuleTest$UsesExternalResource$1	3	3	0	3	5	3
202	org.junit.tests.experimental.rules.ExternalResourceRuleTest$UsesExternalResource	2	1	0	3	5	1
203	org.junit.tests.experimental.rules.ExternalResourceRuleTest	3	1	0	4	12	1
204	org.junit.tests.experimental.rules.MethodRulesTest$1	0	1	0	0	0	0
205	org.junit.tests.experimental.rules.MethodRulesTest$BeforesAndAfters$1	4	2	0	3	6	6
206	org.junit.tests.experimental.rules.MethodRulesTest$BeforesAndAfters	7	1	0	2	13	0
207	org.junit.tests.experimental.rules.MethodRulesTest$CustomTestName$1	2	2	0	3	5	0
208	org.junit.tests.experimental.rules.MethodRulesTest$CustomTestName	2	2	0	4	4	1
209	org.junit.tests.experimental.rules.MethodRulesTest$ExampleTest$1$1	2	2	0	3	5	0
210	org.junit.tests.experimental.rules.MethodRulesTest$ExampleTest$1	2	1	0	5	4	1
211	org.junit.tests.experimental.rules.MethodRulesTest$ExampleTest	2	1	0	2	4	1
212	org.junit.tests.experimental.rules.MethodRulesTest$MultipleRuleTest$Increment$1	2	2	0	3	5	0
213	org.junit.tests.experimental.rules.MethodRulesTest$MultipleRuleTest$Increment	3	1	0	5	5	3

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
214	org.junit.tests.experimental.rules.MethodRulesTest$MultipleRuleTest	2	1	0	3	4	1
215	org.junit.tests.experimental.rules.MethodRulesTest$NoRulesTest	2	1	0	0	3	1
216	org.junit.tests.experimental.rules.MethodRulesTest$OnFailureTest$1	2	2	0	4	10	1
217	org.junit.tests.experimental.rules.MethodRulesTest$OnFailureTest	2	1	0	3	5	1
218	org.junit.tests.experimental.rules.MethodRulesTest$PrivateRule	2	1	0	2	4	1
219	org.junit.tests.experimental.rules.MethodRulesTest$SonOfExampleTest	1	0	0	1	2	0
220	org.junit.tests.experimental.rules.MethodRulesTest$SonOfWrongTypedField	1	0	0	1	2	0
221	org.junit.tests.experimental.rules.MethodRulesTest$UsesCustomMethodRule	2	1	0	2	5	0
222	org.junit.tests.experimental.rules.MethodRulesTest$WatchmanTest$1	3	2	0	3	11	3
223	org.junit.tests.experimental.rules.MethodRulesTest$WatchmanTest	6	1	0	3	13	9
224	org.junit.tests.experimental.rules.MethodRulesTest$WrongTypedField	2	1	0	0	3	1
225	org.junit.tests.experimental.rules.MethodRulesTest	15	1	0	10	35	95
226	org.junit.tests.experimental.rules.NameRulesTest$BeforeAndAfterTest	4	1	0	2	8	0
227	org.junit.tests.experimental.rules.NameRulesTest$TestNames	3	1	0	2	7	0
228	org.junit.tests.experimental.rules.NameRulesTest	1	1	0	0	2	0
229	org.junit.tests.experimental.rules.TempFolderRuleTest$CreatesSubFolder	2	1	0	3	11	0
230	org.junit.tests.experimental.rules.TempFolderRuleTest$HasTempFolder	2	1	0	3	9	0
231	org.junit.tests.experimental.rules.TempFolderRuleTest	7	1	0	5	18	9
232	org.junit.tests.experimental.rules.TestRuleTest$1	0	1	0	0	0	0
233	org.junit.tests.experimental.rules.TestRuleTest$BeforesAndAfters$1	4	3	0	3	6	6

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
234	org.junit.tests.experimental.rules.TestRuleTest$BeforesAndAfters	7	1	0	2	13	0
235	org.junit.tests.experimental.rules.TestRuleTest$BothKindsOfRule	3	2	0	5	4	0
236	org.junit.tests.experimental.rules.TestRuleTest$CustomTestName$1	2	2	0	3	5	0
237	org.junit.tests.experimental.rules.TestRuleTest$CustomTestName	2	2	0	4	4	1
238	org.junit.tests.experimental.rules.TestRuleTest$ExampleTest$1$1	2	2	0	3	5	0
239	org.junit.tests.experimental.rules.TestRuleTest$ExampleTest$1	2	2	0	5	4	1
240	org.junit.tests.experimental.rules.TestRuleTest$ExampleTest	2	1	0	2	4	1
241	org.junit.tests.experimental.rules.TestRuleTest$MultipleRuleTest$Increment$1	2	2	0	3	5	0
242	org.junit.tests.experimental.rules.TestRuleTest$MultipleRuleTest$Increment	3	2	0	5	5	3
243	org.junit.tests.experimental.rules.TestRuleTest$MultipleRuleTest	2	1	0	3	4	1
244	org.junit.tests.experimental.rules.TestRuleTest$NoRulesTest	2	1	0	0	3	1
245	org.junit.tests.experimental.rules.TestRuleTest$OneFieldTwoKindsOfRule	2	1	0	2	5	0
246	org.junit.tests.experimental.rules.TestRuleTest$OnFailureTest$1	2	3	0	4	10	1
247	org.junit.tests.experimental.rules.TestRuleTest$OnFailureTest	2	1	0	3	5	1
248	org.junit.tests.experimental.rules.TestRuleTest$PrivateRule	2	1	0	2	4	1
249	org.junit.tests.experimental.rules.TestRuleTest$SonOfExampleTest	1	0	0	1	2	0
250	org.junit.tests.experimental.rules.TestRuleTest$SonOfWrongTypedField	1	0	0	1	2	0
251	org.junit.tests.experimental.rules.TestRuleTest$UsesCustomMethodRule	2	1	0	2	5	0
252	org.junit.tests.experimental.rules.TestRuleTest$WatchmanTest$1	3	3	0	3	11	3
253	org.junit.tests.experimental.rules.TestRuleTest$WatchmanTest	6	1	0	3	13	9

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
254	org.junit.tests.experimental.rules.TestRuleTest$WrongTypedField	2	1	0	0	3	1
255	org.junit.tests.experimental.rules.TestRuleTest	16	1	0	10	39	110
256	org.junit.tests.experimental.rules.TimeoutRuleTest$HasGlobalTimeout	3	1	0	2	8	1
257	org.junit.tests.experimental.rules.TimeoutRuleTest	2	1	0	6	8	1
258	org.junit.tests.experimental.rules.VerifierRuleTest$UsesErrorCollector	2	1	0	1	6	0
259	org.junit.tests.experimental.rules.VerifierRuleTest$UsesErrorCollectorCheckSucceeds$1	2	1	0	1	4	1
260	org.junit.tests.experimental.rules.VerifierRuleTest$UsesErrorCollectorCheckSucceeds$2	2	1	0	1	4	1
261	org.junit.tests.experimental.rules.VerifierRuleTest$UsesErrorCollectorCheckSucceeds	2	1	0	3	7	0
262	org.junit.tests.experimental.rules.VerifierRuleTest$UsesErrorCollectorCheckSucceedsPasses$1	2	1	0	1	4	1
263	org.junit.tests.experimental.rules.VerifierRuleTest$UsesErrorCollectorCheckSucceedsPasses	2	1	0	3	8	0
264	org.junit.tests.experimental.rules.VerifierRuleTest$UsesErrorCollectorCheckThat	2	1	0	3	7	0
265	org.junit.tests.experimental.rules.VerifierRuleTest$UsesErrorCollectorTwice	2	1	0	1	6	0
266	org.junit.tests.experimental.rules.VerifierRuleTest$UsesVerifier$1	2	3	0	3	4	1
267	org.junit.tests.experimental.rules.VerifierRuleTest$UsesVerifier	2	1	0	3	5	1
268	org.junit.tests.experimental.rules.VerifierRuleTest	8	1	0	4	17	28
269	org.junit.tests.experimental.theories.AllMembersSupplierTest$HasDataPoints	2	1	0	0	4	1
270	org.junit.tests.experimental.theories.AllMembersSupplierTest	2	1	0	6	13	1

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
271	org.junit.tests.experimental.theories.ParameterizedAssertionErrorTest$1	2	1	0	1	4	1
272	org.junit.tests.experimental.theories.ParameterizedAssertionErrorTest	7	1	0	7	19	21
273	org.junit.tests.experimental.theories.ParameterSignatureTest	6	1	0	6	19	15
274	org.junit.tests.experimental.theories.extendingwithstubs.Correspondent	1	1	0	0	1	0
275	org.junit.tests.experimental.theories.extendingwithstubs.Guesser$1	4	2	0	4	11	4
276	org.junit.tests.experimental.theories.extendingwithstubs.Guesser$GuessMap	6	3	0	1	27	15
277	org.junit.tests.experimental.theories.extendingwithstubs.Guesser	8	0	0	6	20	16
278	org.junit.tests.experimental.theories.extendingwithstubs.GuesserQueue$ReguessableDecorator	4	0	0	4	8	0
279	org.junit.tests.experimental.theories.extendingwithstubs.GuesserQueue	5	4	0	4	13	8
280	org.junit.tests.experimental.theories.extendingwithstubs.MethodCall	8	1	0	1	21	6
281	org.junit.tests.experimental.theories.extendingwithstubs.ReguessableValue	2	2	0	2	3	1
282	org.junit.tests.experimental.theories.extendingwithstubs.StringableObject	4	1	0	0	11	0
283	org.junit.tests.experimental.theories.extendingwithstubs.Stub	0	1	0	0	0	0
284	org.junit.tests.experimental.theories.extendingwithstubs.StubbedTheories$StubbedTheoryAnchor	4	3	0	10	25	0
285	org.junit.tests.experimental.theories.extendingwithstubs.StubbedTheories	2	5	0	6	5	1
286	org.junit.tests.experimental.theories.extendingwithstubs.StubbedTheoriesTest	2	1	0	4	6	1
287	org.junit.tests.experimental.theories.runner.SuccessfulWithDataPointFields$BeforeAndAfterEachTime	6	1	0	1	8	3

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
288	org.junit.tests.experimental.theories.runner.SuccessfulWithDataPointFields$BeforeAndAfterOnSameInstance	4	1	0	1	6	0
289	org.junit.tests.experimental.theories.runner.SuccessfulWithDataPointFields$DifferentTypesInConstructor	2	1	0	0	3	1
290	org.junit.tests.experimental.theories.runner.SuccessfulWithDataPointFields$HasATwoParameterTheory	3	1	0	3	7	3
291	org.junit.tests.experimental.theories.runner.SuccessfulWithDataPointFields$NewObjectEachTime	3	1	0	3	10	1
292	org.junit.tests.experimental.theories.runner.SuccessfulWithDataPointFields$OneTestTwoAnnotations	5	1	0	1	7	0
293	org.junit.tests.experimental.theories.runner.SuccessfulWithDataPointFields$PositiveInts	2	1	0	2	5	0
294	org.junit.tests.experimental.theories.runner.SuccessfulWithDataPointFields$PositiveIntsWithMethodParams	2	1	0	2	5	0
295	org.junit.tests.experimental.theories.runner.SuccessfulWithDataPointFields$PositiveIntsWithNegativeField	2	1	0	2	5	0
296	org.junit.tests.experimental.theories.runner.SuccessfulWithDataPointFields$StaticPublicNonDataPoints	3	1	0	1	5	3
297	org.junit.tests.experimental.theories.runner.SuccessfulWithDataPointFields	1	1	0	0	2	0
298	org.junit.tests.experimental.theories.runner.TheoriesPerformanceTest$UpToTen	3	1	0	0	4	3
299	org.junit.tests.experimental.theories.runner.TheoriesPerformanceTest	2	1	0	5	7	1
300	org.junit.tests.experimental.theories.runner.UnsuccessfulWithDataPointFields$DataPointsMustBeStatic	2	1	0	0	3	1
301	org.junit.tests.experimental.theories.runner.UnsuccessfulWithDataPointFields$DoesntUseParams	3	1	0	3	7	3

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
302	org.junit.tests.experimental.theories.runner.UnsuccessfulWithDataPointFields$HasATheory	3	1	0	3	7	3
303	org.junit.tests.experimental.theories.runner.UnsuccessfulWithDataPointFields$NullsOK	3	1	0	3	6	3
304	org.junit.tests.experimental.theories.runner.UnsuccessfulWithDataPointFields$TheoriesMustBePublic	3	1	0	0	4	3
305	org.junit.tests.experimental.theories.runner.UnsuccessfulWithDataPointFields	8	1	0	10	25	28
306	org.junit.tests.experimental.theories.runner.WhenNoParametersMatch$AssumptionsFail	3	1	0	2	6	1
307	org.junit.tests.experimental.theories.runner.WhenNoParametersMatch	3	1	0	7	18	3
308	org.junit.tests.experimental.theories.runner.WithDataPointMethod$DataPointMethodReturnsMutableObject	4	1	0	3	11	6
309	org.junit.tests.experimental.theories.runner.WithDataPointMethod$HasDataPointMethod	3	1	0	0	4	3
310	org.junit.tests.experimental.theories.runner.WithDataPointMethod$HasDateMethod	6	1	0	0	8	15
311	org.junit.tests.experimental.theories.runner.WithDataPointMethod$HasUglyDataPointMethod	4	1	0	0	6	6
312	org.junit.tests.experimental.theories.runner.WithDataPointMethod	9	1	0	11	27	36
313	org.junit.tests.experimental.theories.runner.WithExtendedParameterSources$DataPointArrayMethod	4	1	0	0	8	4
314	org.junit.tests.experimental.theories.runner.WithExtendedParameterSources$DataPointArrays	3	1	0	0	7	1
315	org.junit.tests.experimental.theories.runner.WithExtendedParameterSources$DataPointArrayToBeUsedForWholeParameter	3	1	0	0	7	1

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
316	org.junit.tests.experimental.theories.runner.WithExtendedParameterSources$DataPointMalformedArrayMethods	6	1	0	0	10	13
317	org.junit.tests.experimental.theories.runner.WithExtendedParameterSources$ParameterAnnotations	2	1	0	3	6	1
318	org.junit.tests.experimental.theories.runner.WithExtendedParameterSources$ShouldFilterNull	3	1	0	3	6	3
319	org.junit.tests.experimental.theories.runner.WithExtendedParameterSources	7	1	0	10	13	21
320	org.junit.tests.experimental.theories.runner.WithOnlyTestAnnotations$ErrorWhenTestHasParametersDespiteTheories	3	1	0	0	4	3
321	org.junit.tests.experimental.theories.runner.WithOnlyTestAnnotations$HonorExpectedException	2	1	0	0	3	1
322	org.junit.tests.experimental.theories.runner.WithOnlyTestAnnotations$HonorExpectedExceptionPasses	2	1	0	0	4	1
323	org.junit.tests.experimental.theories.runner.WithOnlyTestAnnotations$HonorTimeout	2	1	0	0	4	1
324	org.junit.tests.experimental.theories.runner.WithOnlyTestAnnotations	5	1	0	9	21	10
325	org.junit.tests.junit3compatibility.AllTestsTest$All	2	1	0	2	5	1
326	org.junit.tests.junit3compatibility.AllTestsTest$AllJUnit4	2	1	0	3	6	1
327	org.junit.tests.junit3compatibility.AllTestsTest$BadSuiteMethod	2	1	0	1	4	1
328	org.junit.tests.junit3compatibility.AllTestsTest$JUnit4Test	2	1	0	1	4	1
329	org.junit.tests.junit3compatibility.AllTestsTest$OneTest	2	3	0	2	4	1
330	org.junit.tests.junit3compatibility.AllTestsTest	7	1	0	7	18	19
331	org.junit.tests.junit3compatibility.ClassRequestTest$PrivateSuiteMethod	2	1	0	1	3	1

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
332	org.junit.tests.junit3compatibility.ClassRequestTest	2	1	0	3	6	1
333	org.junit.tests.junit3compatibility.ForwardCompatibilityPrintingTest$1	2	0	0	3	5	1
334	org.junit.tests.junit3compatibility.ForwardCompatibilityPrintingTest$2	2	3	0	2	4	1
335	org.junit.tests.junit3compatibility.ForwardCompatibilityPrintingTest$3	2	0	0	3	5	1
336	org.junit.tests.junit3compatibility.ForwardCompatibilityPrintingTest$ATest	2	1	0	1	4	1
337	org.junit.tests.junit3compatibility.ForwardCompatibilityPrintingTest$TestResultPrinter	2	2	0	1	3	1
338	org.junit.tests.junit3compatibility.ForwardCompatibilityPrintingTest	4	3	0	11	21	6
339	org.junit.tests.junit3compatibility.ForwardCompatibilityTest$1	5	1	0	4	11	0
340	org.junit.tests.junit3compatibility.ForwardCompatibilityTest$BeforeClassTest	7	1	0	1	11	21
341	org.junit.tests.junit3compatibility.ForwardCompatibilityTest$ErrorTest	2	1	0	1	3	1
342	org.junit.tests.junit3compatibility.ForwardCompatibilityTest$ExceptionInBeforeTest	3	1	0	0	5	3
343	org.junit.tests.junit3compatibility.ForwardCompatibilityTest$ExpectedTest	2	1	0	0	4	1
344	org.junit.tests.junit3compatibility.ForwardCompatibilityTest$InvalidMethodTest	3	1	0	0	4	3
345	org.junit.tests.junit3compatibility.ForwardCompatibilityTest$MarkerRunner	4	2	0	4	6	6
346	org.junit.tests.junit3compatibility.ForwardCompatibilityTest$NewTest	4	1	0	1	8	6
347	org.junit.tests.junit3compatibility.ForwardCompatibilityTest$NoExceptionTest	2	1	0	0	3	1
348	org.junit.tests.junit3compatibility.ForwardCompatibilityTest$NoTests	1	1	0	0	2	0

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
349	org.junit.tests.junit3compatibility.ForwardCompatibilityTest$UnExpectedExceptionTest	2	1	0	0	4	1
350	org.junit.tests.junit3compatibility.ForwardCompatibilityTest	15	3	0	7	43	97
351	org.junit.tests.junit3compatibility.InitializationErrorForwardCompatibilityTest$	1	0	1	0	0	0
352	org.junit.tests.junit3compatibility.InitializationErrorForwardCompatibilityTest$CantInitialize	3	2	0	3	5	3
353	org.junit.tests.junit3compatibility.InitializationErrorForwardCompatibilityTest$CantInitializeTests	1	1	0	0	2	0
354	org.junit.tests.junit3compatibility.InitializationErrorForwardCompatibilityTest$ErrorRememberingListener	7	1	0	5	8	19
355	org.junit.tests.junit3compatibility.InitializationErrorForwardCompatibilityTest$InitializesWithError	1	4	0	1	3	0
356	org.junit.tests.junit3compatibility.InitializationErrorForwardCompatibilityTest	5	1	0	8	23	0
357	org.junit.tests.junit3compatibility.JUnit38ClassRunnerTest$1	2	2	0	4	5	1
358	org.junit.tests.junit3compatibility.JUnit38ClassRunnerTest$AnnotatedTest	2	1	0	1	4	1
359	org.junit.tests.junit3compatibility.JUnit38ClassRunnerTest$ClassWithInvalidMethod	2	3	0	1	3	1
360	org.junit.tests.junit3compatibility.JUnit38ClassRunnerTest$MyTest	2	3	0	1	3	1
361	org.junit.tests.junit3compatibility.JUnit38ClassRunnerTest$OneTest	2	3	0	1	3	1
362	org.junit.tests.junit3compatibility.JUnit38ClassRunnerTest	5	1	0	13	26	10
363	org.junit.tests.junit3compatibility.OldTestClassAdaptingListenerTest$1	1	3	0	2	2	0
364	org.junit.tests.junit3compatibility.OldTestClassAdaptingListenerTest	2	1	0	9	14	1
365	org.junit.tests.junit3compatibility.OldTests	2	1	0	2	4	1

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
366	org.junit.tests.junit3compatibility.SuiteMethodTest$CompatibilityTest	3	1	0	2	5	3
367	org.junit.tests.junit3compatibility.SuiteMethodTest$NewTest	3	1	0	3	5	3
368	org.junit.tests.junit3compatibility.SuiteMethodTest$NewTestSuiteFails	3	1	0	3	5	3
369	org.junit.tests.junit3compatibility.SuiteMethodTest$NewTestSuiteNotUsed	6	1	0	3	8	9
370	org.junit.tests.junit3compatibility.SuiteMethodTest$OldTest	3	3	0	4	6	3
371	org.junit.tests.junit3compatibility.SuiteMethodTest	6	1	0	7	21	3
372	org.junit.tests.listening.ListenerTest$1	2	2	0	3	4	1
373	org.junit.tests.listening.ListenerTest$2	2	2	0	3	4	1
374	org.junit.tests.listening.ListenerTest$OneTest	2	1	0	0	3	1
375	org.junit.tests.listening.ListenerTest	3	1	0	6	14	1
376	org.junit.tests.listening.RunnerTest$1	2	0	0	3	4	0
377	org.junit.tests.listening.RunnerTest$Example	2	1	0	0	3	1
378	org.junit.tests.listening.RunnerTest$ExampleTest	2	3	0	1	3	1
379	org.junit.tests.listening.RunnerTest$MyListener	2	2	0	3	4	1
380	org.junit.tests.listening.RunnerTest$NewExample	2	1	0	0	3	1
381	org.junit.tests.listening.RunnerTest	5	1	0	6	13	8
382	org.junit.tests.listening.TestListenerTest$ErrorListener	2	2	0	3	4	1
383	org.junit.tests.listening.TestListenerTest$ExceptionListener	2	0	0	3	4	0
384	org.junit.tests.listening.TestListenerTest$OneTest	2	1	0	0	3	1
385	org.junit.tests.listening.TestListenerTest	4	1	0	8	17	6
386	org.junit.tests.listening.TextListenerTest$ErrorTest	2	1	0	0	4	1
387	org.junit.tests.listening.TextListenerTest$OneTest	2	1	0	0	3	1

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
388	org.junit.tests.listening.TextListenerTest$Slow	2	1	0	0	4	1
389	org.junit.tests.listening.TextListenerTest	6	3	0	7	24	3
390	org.junit.tests.listening.UserStopTest$OneTest	2	1	0	0	3	1
391	org.junit.tests.listening.UserStopTest	4	1	0	4	11	0
392	org.junit.tests.manipulation.FilterableTest$FilteredRunner$1	3	2	0	3	6	3
393	org.junit.tests.manipulation.FilterableTest$FilteredRunner	1	5	0	3	4	0
394	org.junit.tests.manipulation.FilterableTest$FilteredTest	4	1	0	1	8	6
395	org.junit.tests.manipulation.FilterableTest	2	1	0	3	6	1
396	org.junit.tests.manipulation.SingleMethodTest$1	3	2	0	3	4	3
397	org.junit.tests.manipulation.SingleMethodTest$HasSuiteMethod	4	1	0	2	6	6
398	org.junit.tests.manipulation.SingleMethodTest$OneTimeSetup	4	1	0	1	5	6
399	org.junit.tests.manipulation.SingleMethodTest$OneTwoSuite	1	1	0	0	2	0
400	org.junit.tests.manipulation.SingleMethodTest$ParameterizedOneTimeBeforeClass	4	1	0	1	7	6
401	org.junit.tests.manipulation.SingleMethodTest$ParameterizedOneTimeSetup	3	1	0	0	6	3
402	org.junit.tests.manipulation.SingleMethodTest$TestOne	3	1	0	0	4	3
403	org.junit.tests.manipulation.SingleMethodTest$TestTwo	3	1	0	0	4	3
404	org.junit.tests.manipulation.SingleMethodTest	9	1	0	12	32	30
405	org.junit.tests.manipulation.SortableTest$1	3	1	0	1	6	3
406	org.junit.tests.manipulation.SortableTest$2	3	1	0	1	6	3
407	org.junit.tests.manipulation.SortableTest$TestClassRunnerIsSortable$Enclosing$A	4	1	0	1	6	6

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
408	org.junit.tests.manipulation.SortableTest$TestClassRunnerIsSortable$Enclosing$B	4	1	0	1	6	6
409	org.junit.tests.manipulation.SortableTest$TestClassRunnerIsSortable$Enclosing	1	1	0	0	2	0
410	org.junit.tests.manipulation.SortableTest$TestClassRunnerIsSortable$SortMe	4	1	0	1	6	6
411	org.junit.tests.manipulation.SortableTest$TestClassRunnerIsSortable	8	1	0	5	20	0
412	org.junit.tests.manipulation.SortableTest$TestClassRunnerIsSortableWithSuiteMethod$SortMe	5	1	0	3	8	10
413	org.junit.tests.manipulation.SortableTest$TestClassRunnerIsSortableWithSuiteMethod	6	1	0	5	18	0
414	org.junit.tests.manipulation.SortableTest$UnsortableRunnersAreHandledWithoutCrashing$Unsortable	2	1	0	0	3	1
415	org.junit.tests.manipulation.SortableTest$UnsortableRunnersAreHandledWithoutCrashing$UnsortableRunner	3	2	0	3	4	3
416	org.junit.tests.manipulation.SortableTest$UnsortableRunnersAreHandledWithoutCrashing	2	1	0	4	8	1
417	org.junit.tests.manipulation.SortableTest	5	1	0	2	8	10
418	org.junit.tests.running.classes.EnclosedTest$Enclosing$A	3	1	0	0	4	3
419	org.junit.tests.running.classes.EnclosedTest$Enclosing$B	4	1	0	0	5	6
420	org.junit.tests.running.classes.EnclosedTest$Enclosing	1	1	0	0	2	0
421	org.junit.tests.running.classes.EnclosedTest	4	1	0	6	15	6
422	org.junit.tests.running.classes.IgnoreClassTest$IgnoreMe	3	1	0	1	5	3
423	org.junit.tests.running.classes.IgnoreClassTest	2	1	0	3	7	1
424	org.junit.tests.running.classes.ParameterizedTestTest$BeforeAndAfter	5	1	0	1	9	1

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
425	org.junit.tests.running.classes.ParameterizedTestTest$EmptyTest	3	1	0	1	5	3
426	org.junit.tests.running.classes.ParameterizedTestTest$FibonacciTest	4	1	0	1	8	4
427	org.junit.tests.running.classes.ParameterizedTestTest$IncorrectTest	3	1	0	0	6	3
428	org.junit.tests.running.classes.ParameterizedTestTest$PrivateConstructor	3	1	0	0	6	3
429	org.junit.tests.running.classes.ParameterizedTestTest$ProtectedParametersTest	3	1	0	0	5	3
430	org.junit.tests.running.classes.ParameterizedTestTest$WrongElementType	3	1	0	0	5	3
431	org.junit.tests.running.classes.ParameterizedTestTest	12	1	0	11	40	64
432	org.junit.tests.running.classes.RunWithTest$BadRunner	3	2	0	3	4	3
433	org.junit.tests.running.classes.RunWithTest$Empty	1	1	0	0	2	0
434	org.junit.tests.running.classes.RunWithTest$ExampleRunner	4	2	0	4	7	6
435	org.junit.tests.running.classes.RunWithTest$ExampleTest	1	1	0	0	2	0
436	org.junit.tests.running.classes.RunWithTest$SubExampleTest	1	0	0	1	2	0
437	org.junit.tests.running.classes.RunWithTest	5	1	0	4	17	4
438	org.junit.tests.running.classes.SuiteTest$All	1	1	0	0	2	0
439	org.junit.tests.running.classes.SuiteTest$AllWithBeforeAndAfterClass	3	1	0	1	5	3
440	org.junit.tests.running.classes.SuiteTest$AllWithOutAnnotation	1	1	0	0	2	0
441	org.junit.tests.running.classes.SuiteTest$BiInfiniteLoop	1	1	0	0	2	0
442	org.junit.tests.running.classes.SuiteTest$Hercules	1	1	0	0	2	0
443	org.junit.tests.running.classes.SuiteTest$Hydra	1	1	0	0	2	0
444	org.junit.tests.running.classes.SuiteTest$InfiniteLoop	1	1	0	0	2	0

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
445	org.junit.tests.running.classes.SuiteTest$InheritsAll	1	0	0	1	2	0
446	org.junit.tests.running.classes.SuiteTest$NoSuiteClassesAnnotation	1	1	0	1	2	0
447	org.junit.tests.running.classes.SuiteTest$TestA	2	1	0	0	3	1
448	org.junit.tests.running.classes.SuiteTest$TestB	2	1	0	1	4	1
449	org.junit.tests.running.classes.SuiteTest$WithoutDefaultConstructor	1	1	0	1	2	0
450	org.junit.tests.running.classes.SuiteTest	16	1	0	12	48	114
451	org.junit.tests.running.classes.TestClassTest$ManyMethods	11	1	0	0	12	55
452	org.junit.tests.running.classes.TestClassTest$SubclassWithField	1	0	0	2	2	0
453	org.junit.tests.running.classes.TestClassTest$SuperclassWithField	1	1	0	1	2	0
454	org.junit.tests.running.classes.TestClassTest$TwoConstructors	2	1	0	0	3	1
455	org.junit.tests.running.classes.TestClassTest	3	1	0	4	10	3
456	org.junit.tests.running.classes.UseSuiteAsASuperclassTest$AllWithMySuite	1	1	0	0	2	0
457	org.junit.tests.running.classes.UseSuiteAsASuperclassTest$MySuite	1	4	0	2	2	0
458	org.junit.tests.running.classes.UseSuiteAsASuperclassTest$TestA	2	1	0	0	3	1
459	org.junit.tests.running.classes.UseSuiteAsASuperclassTest$TestB	2	1	0	1	4	1
460	org.junit.tests.running.classes.UseSuiteAsASuperclassTest	2	1	0	3	8	1
461	org.junit.tests.running.core.CommandLineTest$Count	2	1	0	1	4	1
462	org.junit.tests.running.core.CommandLineTest$Example	2	1	0	1	4	1
463	org.junit.tests.running.core.CommandLineTest	9	1	0	5	19	22
464	org.junit.tests.running.core.JUnitCoreReturnsCorrectExitCodeTest$Fail	2	1	0	1	4	1
465	org.junit.tests.running.core.JUnitCoreReturnsCorrectExitCodeTest$Succeed	2	1	0	0	3	1

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
466	org.junit.tests.running.core.JUnitCoreReturnsCorrectExitCodeTest	5	1	0	4	14	10
467	org.junit.tests.running.core.SystemExitTest$Exit	2	1	0	0	4	1
468	org.junit.tests.running.core.SystemExitTest	2	1	0	1	18	1
469	org.junit.tests.running.methods.AnnotationTest$ErrorInAfterClass	3	1	0	1	5	3
470	org.junit.tests.running.methods.AnnotationTest$ErrorInBeforeClass	3	1	0	1	5	3
471	org.junit.tests.running.methods.AnnotationTest$ExceptionTest	2	1	0	0	4	1
472	org.junit.tests.running.methods.AnnotationTest$FailureTest	2	1	0	1	4	1
473	org.junit.tests.running.methods.AnnotationTest$NoExceptionTest	2	1	0	0	3	1
474	org.junit.tests.running.methods.AnnotationTest$NonStaticOneTimeSetup	3	1	0	0	4	3
475	org.junit.tests.running.methods.AnnotationTest$OldSuiteTest	2	3	0	2	3	1
476	org.junit.tests.running.methods.AnnotationTest$OldTest	2	3	0	2	3	1
477	org.junit.tests.running.methods.AnnotationTest$OneTimeSetup	4	1	0	1	5	6
478	org.junit.tests.running.methods.AnnotationTest$OneTimeTeardown	4	1	0	1	5	6
479	org.junit.tests.running.methods.AnnotationTest$OrderTest	6	1	0	1	10	15
480	org.junit.tests.running.methods.AnnotationTest$RunAllAfterClasses	6	1	0	1	11	15
481	org.junit.tests.running.methods.AnnotationTest$RunAllAfterClassesRegardless	4	1	0	1	9	6
482	org.junit.tests.running.methods.AnnotationTest$RunAllAfters	6	1	0	1	11	15
483	org.junit.tests.running.methods.AnnotationTest$RunAllAftersRegardless	4	1	0	1	9	6
484	org.junit.tests.running.methods.AnnotationTest$SetupFailureTest	3	1	0	1	5	3
485	org.junit.tests.running.methods.AnnotationTest$SetupTest	3	1	0	1	4	3

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
486	org.junit.tests.running.methods.AnnotationTest$SimpleTest	2	1	0	1	3	1
487	org.junit.tests.running.methods.AnnotationTest$SimpleTestWithFutureProofExplicitRunner	2	1	0	1	3	1
488	org.junit.tests.running.methods.AnnotationTest$SubInheritance	6	0	0	2	10	15
489	org.junit.tests.running.methods.AnnotationTest$SubShadowing	4	0	0	2	8	6
490	org.junit.tests.running.methods.AnnotationTest$SubTest	2	0	0	2	6	1
491	org.junit.tests.running.methods.AnnotationTest$SuperInheritance	5	1	0	1	9	10
492	org.junit.tests.running.methods.AnnotationTest$SuperShadowing	3	1	0	1	7	3
493	org.junit.tests.running.methods.AnnotationTest$SuperTest	3	1	0	1	7	3
494	org.junit.tests.running.methods.AnnotationTest$TeardownAfterFailureTest	3	1	0	1	5	3
495	org.junit.tests.running.methods.AnnotationTest$TeardownFailureTest	3	1	0	0	5	3
496	org.junit.tests.running.methods.AnnotationTest$TeardownTest	3	1	0	1	4	3
497	org.junit.tests.running.methods.AnnotationTest$TestAndTeardownFailureTest	3	1	0	0	6	3
498	org.junit.tests.running.methods.AnnotationTest$TwoTests	3	1	0	1	5	3
499	org.junit.tests.running.methods.AnnotationTest	29	3	0	11	57	234
500	org.junit.tests.running.methods.ExpectedTest$Expected	2	1	0	0	4	1
501	org.junit.tests.running.methods.ExpectedTest$ExpectSuperclass	2	1	0	0	4	1
502	org.junit.tests.running.methods.ExpectedTest$NoneThrown	2	1	0	0	3	1
503	org.junit.tests.running.methods.ExpectedTest$Unexpected	2	1	0	0	4	1
504	org.junit.tests.running.methods.ExpectedTest	5	1	0	4	20	10

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
505	org.junit.tests.running.methods.InheritedTestTest$Sub	1	0	0	1	2	0
506	org.junit.tests.running.methods.InheritedTestTest$SubWithBefore	2	0	0	2	4	1
507	org.junit.tests.running.methods.InheritedTestTest$Super	2	1	0	0	3	1
508	org.junit.tests.running.methods.InheritedTestTest	3	1	0	3	8	3
509	org.junit.tests.running.methods.ParameterizedTestMethodTest$EverythingWrong	26	1	0	0	27	325
510	org.junit.tests.running.methods.ParameterizedTestMethodTest$SubShadows	2	0	0	1	3	1
511	org.junit.tests.running.methods.ParameterizedTestMethodTest$SubWrong	2	0	0	1	3	1
512	org.junit.tests.running.methods.ParameterizedTestMethodTest$SuperWrong	2	1	0	0	3	1
513	org.junit.tests.running.methods.ParameterizedTestMethodTest	5	1	0	5	14	8
514	org.junit.tests.running.methods.TestMethodTest$Confused	3	1	0	0	4	3
515	org.junit.tests.running.methods.TestMethodTest$ConstructorParameter	2	1	0	0	3	1
516	org.junit.tests.running.methods.TestMethodTest$EverythingWrong	26	1	0	0	27	325
517	org.junit.tests.running.methods.TestMethodTest$IgnoredTest	4	1	0	0	5	6
518	org.junit.tests.running.methods.TestMethodTest$OnlyTestIsIgnored	2	1	0	0	3	1
519	org.junit.tests.running.methods.TestMethodTest$SubShadows	2	0	0	1	3	1
520	org.junit.tests.running.methods.TestMethodTest$SubWrong	2	0	0	1	3	1
521	org.junit.tests.running.methods.TestMethodTest$SuperWrong	2	1	0	0	3	1
522	org.junit.tests.running.methods.TestMethodTest	10	1	0	7	28	45

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
523	org.junit.tests.running.methods.TimeoutTest$FailureWithTimeoutRunTimeExceptionTest	2	1	0	0	4	1
524	org.junit.tests.running.methods.TimeoutTest$FailureWithTimeoutTest	2	1	0	1	4	1
525	org.junit.tests.running.methods.TimeoutTest$ImpatientLoopTest	3	1	0	0	4	3
526	org.junit.tests.running.methods.TimeoutTest$InfiniteLoopTest	3	1	0	0	5	3
527	org.junit.tests.running.methods.TimeoutTest$SuccessWithTimeoutTest	2	1	0	0	3	1
528	org.junit.tests.running.methods.TimeoutTest$TimeoutFailureTest	2	1	0	0	4	1
529	org.junit.tests.running.methods.TimeoutTest$WillTimeOut	4	1	0	0	6	4
530	org.junit.tests.running.methods.TimeoutTest	11	1	0	10	42	55
531	org.junit.tests.validation.BadlyFormedClassesTest$BadBeforeMethodWithLegacyRunner	3	1	0	0	4	3
532	org.junit.tests.validation.BadlyFormedClassesTest$FaultyConstructor	2	1	0	0	4	1
533	org.junit.tests.validation.BadlyFormedClassesTest$NoTests	1	1	0	0	2	0
534	org.junit.tests.validation.BadlyFormedClassesTest	5	1	0	4	13	10
535	org.junit.tests.validation.FailedConstructionTest$CantConstruct	2	1	0	0	4	1
536	org.junit.tests.validation.FailedConstructionTest	2	1	0	5	9	1
537	org.junit.tests.validation.InaccessibleBaseClassTest	2	1	0	2	4	1
538	org.junit.tests.validation.ValidationTest$NonStaticBeforeClass	3	1	0	0	4	3
539	org.junit.tests.validation.ValidationTest$WrongBeforeClass	2	1	0	0	3	1
540	org.junit.tests.validation.ValidationTest	3	1	0	7	14	3
541	org.junit.tests.validation.anotherpackage.Sub	2	0	0	1	4	1
542	org.junit.tests.validation.anotherpackage.Super	2	1	0	0	3	1

Tabela 18 (continuação): Lista com as métricas de C&K, obtidas com a ferramenta CKJM, para as 563 classes da versão 4.9b2 do *software* jUnit.

Contador	Diretório e Identificação da Classe	WMC	DIT	NOC	CBO	RFC	LCOM
543	org.hamcrest.BaseDescription	13	1	0	4	27	78
544	org.hamcrest.BaseMatcher	3	1	0	3	5	3
545	org.hamcrest.CoreMatchers	21	1	0	11	42	210
546	org.hamcrest.Description	6	1	0	1	6	15
547	org.hamcrest.Factory	0	1	0	0	0	0
548	org.hamcrest.Matcher	2	1	0	1	2	1
549	org.hamcrest.SelfDescribing	1	1	0	1	1	0
550	org.hamcrest.StringDescription	7	2	0	3	14	9
551	org.hamcrest.core.AllOf	5	2	0	3	12	4
552	org.hamcrest.core.AnyOf	5	2	0	3	12	4
553	org.hamcrest.core.DescribedAs	5	2	0	4	20	4
554	org.hamcrest.core.Is	6	2	0	6	12	9
555	org.hamcrest.core.IsAnything	7	2	0	3	9	19
556	org.hamcrest.core.IsEqual	9	2	0	3	16	30
557	org.hamcrest.core.IsInstanceOf	4	2	0	3	8	0
558	org.hamcrest.core.IsNot	5	2	0	5	10	4
559	org.hamcrest.core.IsNull	7	2	0	4	10	21
560	org.hamcrest.core.IsSame	4	2	0	3	7	0
561	org.hamcrest.internal.ArrayIterator	4	1	0	0	11	0
562	org.hamcrest.internal.SelfDescribingValue	2	1	0	2	4	0
563	org.hamcrest.internal.SelfDescribingValueIterator	5	1	0	2	9	0