Indentifying object-oriented software metrics thresholds

Kecia A. M. Ferreira, Mariza A. S. Bigonha, Roberto S. Bigonha, Luiz F. O. Mendes, Heitor C. Almeida, Kecia A. M. Ferreira, Mariza A. S. Bigonha, Roberto S. Bigonha

Dept. Computer Science – Universidade Federal de Minas Gerais (UFMG), Av. Antônio Carlos, 6627 - Pampulha - CEP: 31270-010 - Belo Horizonte - Brazil

Abstract

Despite the importance of software metrics and the large number of proposed metrics, they have not been widely applied in industry yet. One reason for this may be that for most metrics reference values for measurements are not known. This paper presents results of a study on the structure of a large collection of open source software developed in Java, varying size and from different application domains. The aim of this work is the characterization of open source software by means of a set of object-oriented software metrics, namely: LCOM, DIT, coupling factor, afferent couplings, number of public methods and number of public fields. The results of the study provide important insights on the structure of open source software, confirming the intuitive notion that open source software development emphasizes high-quality code. The primary conclusion of this work is the identification of values to be used as reference for the six software metrics. The methodology used in this study can be applied for other software metrics in order to find their reference values.

Keywords:
software metrics thresholds, open source software, object-oriented software

1. Introduction

Software metrics allow measurement, evaluation, control and improvement of software products and processes. Much research has been spent in order to define and evaluate software metrics (Fenton and Neil, 2000; Xenos et al., 2000; Baxter et al., 2006; Kitchenham, 2009). Despite their importance and the large number of proposed metrics, they have not been applied widely

in the industry yet. Tempero (2008) believes that this is due the fact that for most metrics typical values for measurements are not known. If software engineers do not know which values of measures of a software metric can be considered as low, high or acceptable, they are not able to apply it and software metrics will not be very useful. In particular for object-oriented software, little is known about simple information, such as the number of methods in a typical class and the typical number of classes used by a class. Object-oriented software characterization is important in order to obtain this type of information.

The aim of this work is to identify thresholds for a set of object-oriented software metrics. To achieve this goal, we investigated the metrics values most commonly used in practice. We performed a study on the structure of a large collection of open source software developed in Java, from different application domains. The main reason why we studied open source software is the availability of data about this kind of software, including its source code. There is a great amount of software of this type. Sourceforge (www.sourceforge.net), for instance, has more than 176,000 avalialable programs. Furthermore, open source software is thought to have more maintainability than closed source software, and its development seems to solve common problems of traditional software development, since it is possible to produce high-quality software in a brief amount of time (Samoladas et al., 2004). Since maintainability is one of the most important software quality factors and is a condition to open source software success, the characterization of this type of software can bring important insights about its actual structure. The observed characteristics of open source software by means of software metrics in this work reveals that this type of software stresses high quality: classes are low connected each other, have high cohesion, few public methods and fields, and also short inheritance tree. This important insight about open source software leads to consider their measures of metrics as reference.

The paper is organised as follows. Section 2 discusses relevant related work. Section 3 describes the methodology used in this research, providing background of the analysed metrics. Section 4 presents results of this study and their analysis. Section 5 identifies the software metrics thresholds suggested in this work. Section 6 brings the conclusions and suggested future works.

2. Related Work

A large number of metrics have been proposed (Abreu and Carapuça, 1994; Chidamber and Kemerer, 1994; Xenos et al., 2000; Kitchenham, 2009). Despite the amount of effort spent in the definition and evaluation of software metrics, this research area poses great challenges and the typical values of most software metrics are not known yet (Tempero, 2008). Proper metric values interpretation is essential to characterize, evaluate and improve the design of large software, and knowing the typical values of software metrics is necessary in order to interpret the measures of these metrics. Therefore, without knowing metrics thresholds, software community will not be able to apply software metrics in practice (Lanza and Marinescu, 2006). This section discusses works concerned with software structure characterization by means of software metrics and with identification of software metrics thresholds.

Attention has been given by researchers to the way software modules connect with each other. A conclusion drawn by those works is that software seems to be governed by power laws (Baxter et al., 2006; Louridas et al., 2008; Potantin et al., 2005; Puppin and Silvestrini, 2006; Wheeldon and Counsell, 2003). A power law is a probability distribution function in which the probability that a random variable X be equal to x is proportional to a negative power of x, i.e.,$P(X = x) \propto cx^{-k}$. A power law distribution is a heavy-tail distribution. A characteristic of this type of distribution is that the frequency of high values for the random variable is very low and the frequency of low values is high. In this distribution, the mean value is not representative, and so, there is no value that can be considered as a typical value to random variable. A great number of phenomena can be modeled by power laws, for instance: use of word frequency, author citations in papers, phone calls, city populations and some real-world networks, such as in-degree in Internet nodes (Newman, 2003).

Some researches have identified power laws in graphs that represent relationships between classes and objects in an object-oriented system. Potantin et al. (2005) analysed 60 graphs of 35 software systems and concluded that the geometry of relationship between objects, in execution time, is scale free. A scale free graph is different from a graph with edges distributed randomically. In a random graph, mean value of nodes degree is representative, while in a scale free graph this is not true. Wheeldon and Counsell (2003) identified power laws in classes relationship in Java programs. Their study envolved three well-known software systems: JDK (*Java Development Kit*), Apache

Ant and Tomcat, in a total of 6,870 classes. Their work verified power laws in the following types of connections between classes in object-oriented software: inheritance, interface implementation and aggregation, use of a class in parameter lists and use of a class as return type. In addition, power law was verified in the following class characteristics: number of fields, methods and contructors. Louridas et al. (2008) analysed probability distributions that model in and out-degree of software modules. The sample is from programs developed in C, Perl, Java and Ruby. A set of 11 software systems was analysed, among them J2SE SDK, Eclipse, OpenOffice and Ruby. The study concludes that, regardless of the programming paradigm, in and out-degree are governed by power law.

The study of Baxter et al. (2006) investigated the structure of a large number of Java software and it is one of the first studies of this nature. The data set used in their study is from 56 open source software, with varying size and from different application domain. They collected and analysed measures of several metrics and concluded that some metrics fit to a power law and others do not. For example, the study suggests that in-degree distribution and number of subclasses are power law, but out-degree, number of fields and number of public fields are not. This conclusion diverges from findings of the study of Louridas et al. (2008), that found power law in out-degree of classes. Findings of Baxter et al. (2006) and Louridas et al. (2008) are important because they bring information that can allow understanding the shape of open source software. However, the results are not explored in order to identify typical or reference values for the analysed metrics, and the studies did not directly analyse metrics of important quality factors, such as module coupling and cohesion.

Lanza and Marinescu (2006) identify two sources for thresholds values: statistical information and the widely accepted knowledge. Statistics-based thresholds are derived from statistical analisys of data from a population or a sample of a population. Using statistical analysis, they suggest thresholds for three software metrics: number of methods per class, lines of code per method, cyclomatic number per lines of code. They collected these metrics from 37 C++ systems and 45 Java systems. The diversity of size, application domain and type (open-source and commercial software) was the basis of the sample selection. They aim to identify, for those three metrics: the typical values, that include the interval of values from most systems; the lower and the higher values of this interval; and a value which can be considered an outlier, an extreme high value. Considering a normal distribution for the

collected data, they applied average and standard deviation in order to find the thresholds: for each metric, the average is used to define the most typical value and the standard deviation is used to define the two margins of the typical values interval and the outlier. They consider a value as an outlier if it is 50% higher than the highest value of the interval. The methodology applied in their work is useful only if the values follow a normal distribution. However, as pointed by the other studies described in this section, a large number of software metrics follow power law. Then, interpreting these metrics in terms of average values can be extremely misleading.

This paper presents a research that advances characterization of object-oriented open source software by means of six software metrics that have not been studied in this way in previous works: LCOM (lack of cohesion in Methods), DIT (Depth in Inheritance Tree) (Chidamber and Kemerer, 1994), COF (coupling Factor) (Abreu and Carapuça, 1994), afferent couplings, number of public methods and numbeL of public fields. These metrics are described in Section 3.1. This research investigates probability distributions that fit to values of metrics used in the study. Considering the fact that open source software tends to have high-quality (Samoladas et al., 2004), the results are explored in order to identify values that can be taken as reference for measures of those metrics. This is an open question in software engineering and its solution can help providing the effective use of software metrics in software production.

3. Methodology

Data used in this study are from 40 Java open source software, downloaded from SourceForge (*www.sourceforge.net*), varying size from 18 to 3,500 classes, in their latest version up to June 2008. Program codes are from 11 application domains and three types: tool, library and framework. More than 26,000 classes were analysed. Software names, their application domains, types and size are described in Table 1 and 2. A tool, called *Connecta* (Ferreira et al., 2008), was used to collect measures of the metrics. *Connecta* collects measures of object-oriented software metrics from bytecodes of Java programs. For this reason, a criterion to choose software to be analysed in this study was the bytecode availability.

Three types of analysis are made on the collected measures: to the entire data set, by application domain, type and size of software systems. The hypothesis investigated is whether there is a single distribution probability

Table 1: software systems, their application domain, type, size and COF metric

Domain	*Software*	*Type*	*#Classes*	*#Connections*	*COF*
Clustering	Essence	framework	182	543	0,016
	Gridsim	tool	214	774	0,017
	JavaGroups	tool	1061	3807	0,003
	Prevayler	library	90	137	0,017
	Super (Acelet-Scheduler)	tool	246	1085	0,018
Database	DBUnit	framework	289	911	0,011
	ERMaster	tool	569	2187	0,007
	Hibernate	framework	1359	5199	0,003
Desktop	Facilitator	tool	2234	6565	0,001
	Java Gui Builder	tool	60	126	0,036
	Java X11 Library	library	318	1146	0,011
	J-Pilot	tool	142	367	0,018
	Scope	framework	214	535	0,012
Development	Code Generation Library	library	226	662	0,013
	DrJava	tool	2766	9684	0,001
	Find Bugs	tool	1019	3108	0,003
	Jasper Reports	library	1233	5610	0,004
	Junit	framework	154	353	0,015
	Spring	framework	2116	7069	0,002
	BCEL	library	373	2111	0,015

that can model measures of a metric, regardless the application domain and type of software.

This section presents the software metrics analised in this study, the method of fitting data and the approach used to identify reference values.

3.1. Software Metrics

There is a great number of object-oriented software metrics in the literature, among them are highlighted the CK metrics, by Chidamber and Kemerer (1994), and the MOOD set metrics, by Abreu and Carapuça (1994). In this study, the following metrics are used:

COF (Coupling Factor): this metric is calculated for the system. It is based on the concept of *client-server* relationship between classes. Conside-

Table 2: software systems, their application domain, type, size and COF metric

Domain	*Software*	*Type*	*#Classes*	*#Connections*	*COF*
Enterprise	Liferay	framework	14	14	0,077
	Talend	tool	2779	3567	0,000822
	uEngine BPM	framework	708	1774	0,004
	YAWL	tool	382	1186	0,008
Financial	JMoney	tool	193	424	0,019
Games	JSpaceConquest	tool	150	424	0,019
	KoLmafia	tool	810	5106	0,008
	Robocode	tool	213	738	0,016
Hardware	Jcapi	library	21	61	0,145
	LibUSBJava	library	35	90	0,076
	ServoMaster	library	55	117	0,039
Multimedia	CDK	library	3586	14711	0,001
	JPedal	tool	539	1533	0,005
	Pamguard	tool	1503	5267	0,002
Networking	BlueCove	library	142	461	0,023
	DHCP4Java	library	18	29	0,095
	jSLP	library	42	156	0,091
	WiKID Strong Authentication	library	50	27	0,011
Security	JSch	library	110	226	0,022
	OODVS	library	171	325	0,011

ring this concept, a class A is client of a server class B if A references at least one member of B. If A is client of B, then A is connected to B. In this study, when A is a subclass of B, it is also considered that there is a connection from A to B. Therefore, a software system can be modeled as a directed graph. In a software having n classes, the maximum possible number of connections is $n^2 - n$. COF is given by $c/(n^2 - n)$, where c is the number of actual connections in the software. This metric is an indicator of the connectivity level of the system. The higher COF value, the higher the connectivity of the system and the lower its maintainability (Abreu and Carapuça, 1994). As asserted by Meyer (1997), in a software architecture,"every module should communicate with as few others as possible", otherwise changes

and errors may propagate widely in the system.

Number of public fields: this metric is the total number of public fields defined in a class. Using public fields is not a good practice because it favors strong coupling between classes.

Number of public methods: this metric is the total number of public methods defined in a class. It is an indicator of the size of a class.

LCOM (Lack of Cohesion in Methods): this metric, defined by Chidamber and Kemerer (1994), measures the cohesion level of a class by considering the concept of similarity of methods of the class. Two methods are similar if they use at least one common field of their class. LCOM is given by the number of pairs of non-similar methods minus the number of pairs of similar methods. When the number of pairs of non-similar methods is less than the number of pairs of similar methods, LCOM is setted to zero. According to Chidamber and Kemerer (1994), the higher LCOM value, the lower the class cohesion, however a zero value does not necessarily mean good cohesion. There is a large number of cohesion metrics for object-oriented software and LCOM has been critised in the literature (Briand et al., 1999). In spite of this, we consider LCOM in our study because there is no consensual conclusion about the best way on measuring class cohesion. In addition, some cohesion metrics are based on the same idea of similarity used by LCOM.

DIT (Depth of Inheritance Tree): this metric is given by the maximum distance of a class from the root class in the inheritance tree of the system. Inheritance is a powerful technique of software reuse, nevertheless Gamma et al. (1994) claim its immoderate use can make software design more complex. They, then, define a principle: *favor object composition over class inheritance*. In the same vein, Sommerville (2000) argues that inheritance introduces difficulties in the comprehension of objects behavior. An empirical study of Daly et al. (1996) shows that deep inheritance trees makes software maintenance more difficult. DIT indicates how deep is a class in the inheritance tree. It is considered by its authors as an indicator of the design complexity of the system (Chidamber and Kemerer, 1994). The higher the DIT of a class, the more complex its comprehension, because more classes are involved in its analysis.

Afferent couplings: this metric, defined by Ferreira et al. (2008), is based on the same concept of *client-server* relationship between classes used by COF. If A is client of B, then A is connected to B and, so there is an afferent coupling in B from A and a corresponding efferent coupling in A. This metric is the total number of afferent couplings in a class. Classes having high number of afferent couplings play an important role in the system, because errors or modifications on them can widely impact the other ones.

3.2. Data Fitting

A tool, called EasyFit (Mathwave, 2010), was used to make data fitting to various probability distributions, such as Bernoulli, Binomial, Uniform, Geometric, Hypergeometric, Logarithmic, Binomial, Poisson, Normal, t-Student, Chi-square, Exponential, Lognormal, Pareto and Weibull. A probability distribution has two main functions: the *pdf* (*probability density function*), $f(x)$, that indicates the probability the random variable takes a value x, and the *cdf* (*cumulative distribution function*), $F(x)$, that indicates the probability the random variable takes a value less than x. In the experiment of this study, the following probability distributions are well fitted to the data: Poisson and Weibull.

Poisson distribution has *pdf*, $f_p(x)$, and *cdf*, $F_p(x)$, defined by Equations 1 and 2 respectively. The parameter λ of the distribution is the mean value of the random variable.

$$f_p(x) = P(X = x) = \frac{e^{-\lambda}.\lambda^x}{x!} \tag{1}$$

$$F_p(x) = P(X \leq x_0) = \sum_{x=0}^{x=x_0} \frac{e^{-\lambda}.\lambda^x}{x!} \tag{2}$$

Weibull distribution has *pdf*, $f_w(x)$, and *cdf*, $F_w(x)$, with parameters α and β, defined by Equations 3 and 4 respectively. The parameter β is called *scale parameter*. Increasing the value of β has the effect of decreasing the height of the curve and stretching it. The parameter α is called *shape parameter*. If the shape parameter is less than 1, Weibull is a heavy-tail distribution and it can be applied in cases in which the random variable presents left assimetry, i.e., when there is a small number of ocurrences with high values and a far greater number of ocurrences with low values. In this kind of distribution, the mean value is not significant.

$$f_w(x) = P(X = x) = \frac{\alpha}{\beta}(\frac{x}{\beta})^{\alpha-1}e^{-(\frac{x}{\beta})^{\alpha}}, \alpha > 0, \beta > 0 \quad (3)$$

$$F_w(x) = P(X \leq x) = 1 - e^{-(\frac{x}{\beta})^{\alpha}}, \alpha > 0, \beta > 0 \quad (4)$$

3.3. Data Analysis

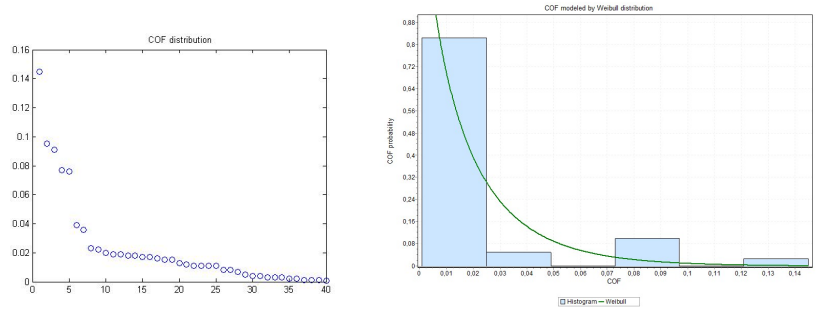
For each metric, data were collected and two plots were genetared: a scatter plot, in order to exhibit the frequency of measures, and the same data in doubly logarithmic scale (log-log scale), in order to observe whether the distribution shows itself to be a power law. When plotted in a log-log scale, power law distributions are right-skewed and have an approximately straight-line form. A power law distribution for a metric indicates that the frequency of high values for the metric is very low and the frequency of low values is high. This analysis brings important insigths on the structure of open source software, by confirming or refuting the intuitive notion that open source software development emphasizes high-quality code. For instance, if LCOM measures appear to have power law distribution, then it is possible conclude that most classes have good cohesion.

Data were fitted to probability distributions and, considering the indication of best fitting from the used tool and the visual analysis of the fittings, it was identified the probability distribution with best fit to the data. If the probabiliy distribution has a representative mean value, like Poisson distribution, this value is taken as typical for the metric, otherwise, we should work with three ranges of metric values: *good*, *regular* and *bad*. The *good* range corresponds to values with high frequency, the *bad* range corresponds to values with probability of ocurrence tending to zero, and the *regular* range is an intermediate one, that corresponds to values that are not too frequent neither have very low frequency.

This aproach was employed based on the following judgment: data are from open source, and development of this kind of software should emphasize high-quality, specially maintainability. Since, open source software systems are thought to be constructed in order to facilitate their maintainance and use, even without documentation or technical supportand, characteristics of this type of software may be taken as a model, and the measures of their metrics can be taken as reference values.

We are not able to confirm that all open source software systems possess high-quality in general. However, some well-known open source software

systems are used in this study, among them JUnit, Hibernate and Spring frameworks, which are widely used in software development industry. We evaluated two software codes qualitatively: BCEL and Robocode. BCEL is a library for manipulation of Java classes bytecodes and we used it in Connecta implementation. We consider BCEL as a well constructed library, modular and easy to use. Robocode is a popular programming game in which Java is used to program robots to do battle against each other. Our evaluation of source code structure of Robocode concludes that its classes are well constructed, most of them do not have public fields and have few and short methods. However, qualitative evaluation of software structure is error-prone, specially for large software systems. Nevertheless, by the results of this study, discussed in Section 4, the sample used shows itself to possess good quality in the sense of the factors evaluated by the metrics studied in this work: classes are low connected each other, have few public fields, short interface, few superclasses and good cohesion. Therefore, we suggest taking the measures most commonly used in practice in open source software as thresholds.



(a) (b)

Figure 1: COF - (a) frequency and (b) fitting to Weibull distribution.

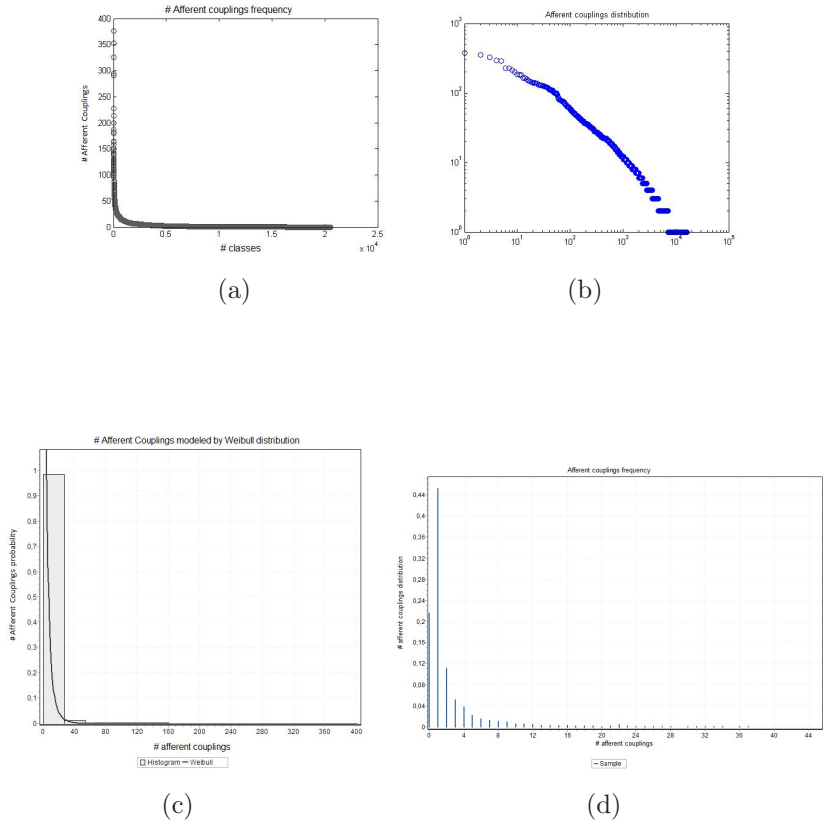


Figure 2: Afferent Couplings - (a) frequency, (b) frequency in log-log scale, (c) fitting to Weibull distribution and (d) frequency detailed.

4. Results

In this section, we describe the findings of our study. First, results of the entire data set are described. Then, we discuss results of the analisys performed on application domains, types and size of the software systems.

4.1. Data Fitting of Metrics in the Entire Data Set

4.1.1. COF

The COF scatter plot in Figure 1a shows that values less than 0,20 are far more frequent than higher values. COF can be modeled by Weibull

distribution, with parameters $\alpha = 0,91927$ and $\beta = 0,01762$. Figure 1b shows fitting data with Weibull distribution. Based on graphic analysis, more than 80% of the programs have COF less than 0,02, the probability that COF takes values 0,02 to 0,14 is quite low, and the probability that COF takes values higher than 0,14 tends to zero. This result points out that, in most cases, open source software is low connected and this is a fact that may contribute to its maintainability.

4.1.2. Afferent Couplings

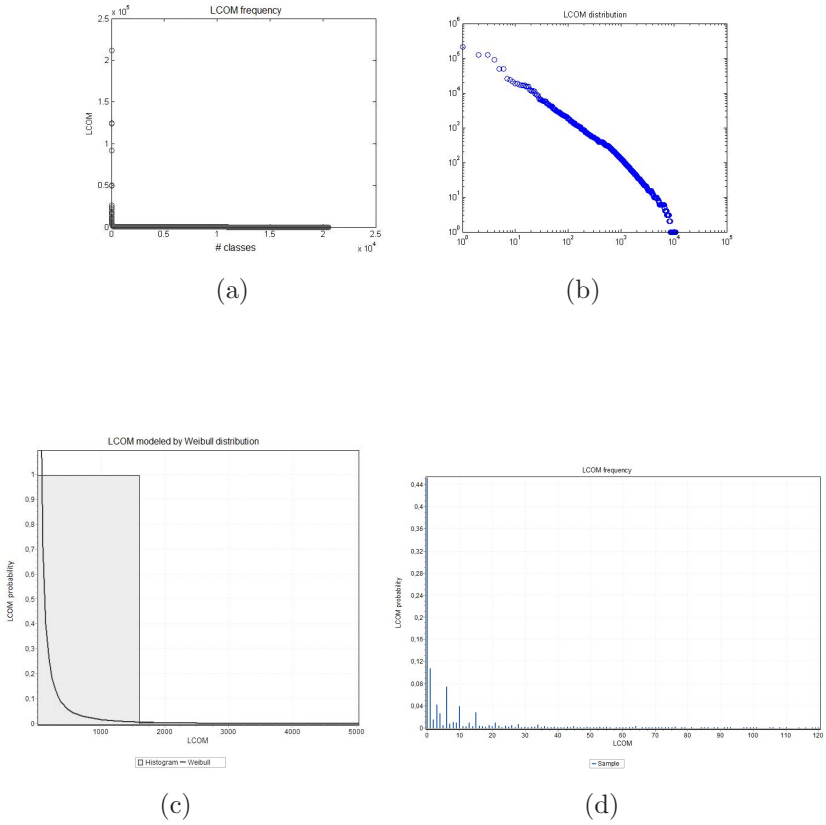
The scatter plot for afferent couplings, shown in Figure 2a, suggests a heavy-tail distribution. Figure 2b shows the same data plotted in logarithmic scale (log-log scale). In this plot, distribution shows itself to be linear, that is the characteristic signature of a power law. There is a small number of classes with high number of afferent couplings and a far higher number of classes with few afferent couplings. As shown by Figure 2c, values of this metric can be modeled by Weibull distribution, with parameters $\alpha = 0,78986$ and $\beta = 3,2228$. Afferent couplings distribution is detailed in Figure 2d. Almost 50% of classes have one afferent coupling at most, the probability that a class takes 1 to 20 afferent couplings is low, and the probability to be greater than 20 tends to zero. This indicates that most classes directly impact only one class at most. This can contribute to maintainabilty and to software quality in general, because a modification or an error in a class would impact in a low number of classes.

4.1.3. LCOM

LCOM also is fitted by a heavy tail distribution. Figure 3a shows a scatter plot of the data set, and Figure 3b shows the same data plotted in log-log scale. This plot indicates that LCOM values follows a power law. Values of LCOM can be modeled by Weibull distribution, as shown in Figure 3c, with parameters $\alpha = 0,23802$ and $\beta = 1,465$. Figure 3d details LCOM distribution. Almost 50% of classes have LCOM equals to zero, that means good cohesion. There are classes with LCOM between 0 and 20 in a low frequency, less than 12%, and the probability that a class has LCOM greater than 20 tends to zero.

4.1.4. DIT

The scatter plot in Figure 4a shows distribution of DIT values and Figure 4b shows the same data in a log-log scale. These plots do not suggest power

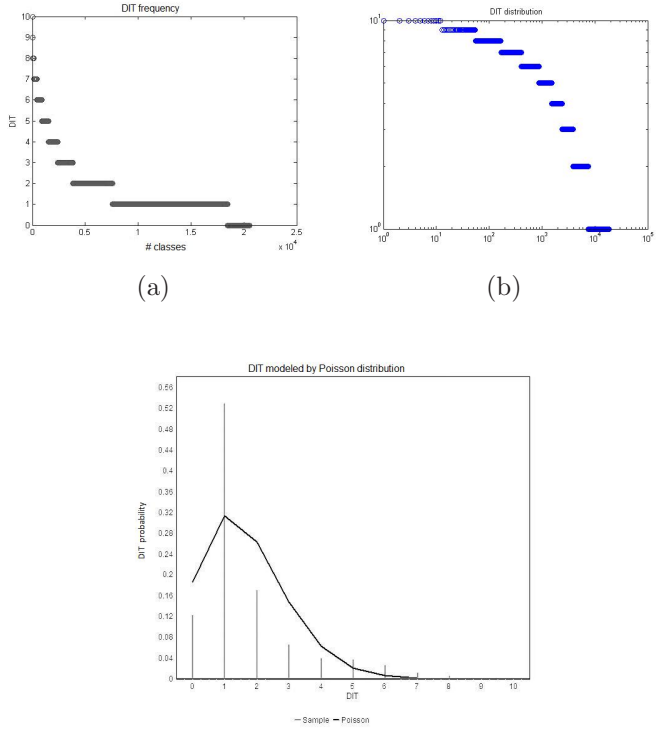


(a) (b)

(c) (d)

Figure 3: LCOM (a) frequency, (b) frequency in log-log scale, (c) fitting to Weibull distribution and (d) frequency detailed.

law characteristics in DIT values distribution. DIT values can be fitted to Poisson distribution, as shown in Figure 4c, with parameters $\lambda = 1,6818$. In Poisson distribution, λ gives the mean value of the random variable. By this finding, in an open source software, the largest distance from a class to the root in the inheritance tree is 2, in general. This reflects that this kind of software do not have very deep inheritance tree, which also contributes to software maintainability by decreseasing software complexity, as asserted by Gamma et al. (1994), Daly et al. (1996) and Sommerville (2000).



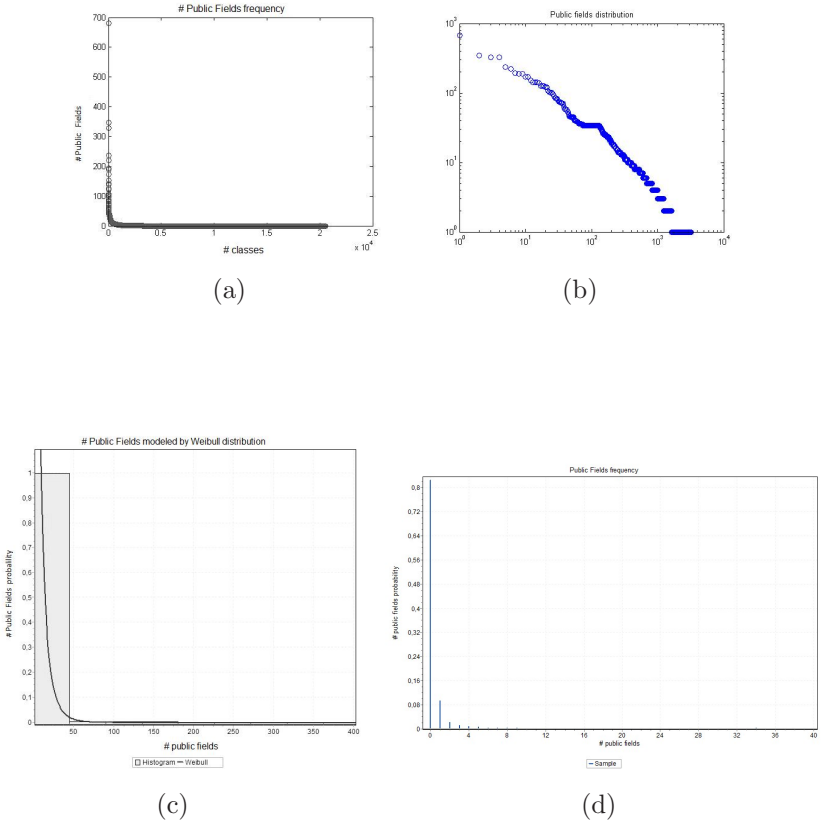
(a) (b)

(c)

Figure 4: DIT - (a) frequency, (b) frequency in log-log scale, (c) fitting to Poisson distribution.

4.1.5. Public Fields

The scatter plot of number of public fields, shown in Figure 5a, reveals that there is a low number of classes with a great number of public fields and, in most cases, this number is near to zero. Figure 5b shows the data plotted in a log-log scale, which indicates that number of public fields in classes also follows a power law.This metric can be modeled by Weibull distribution with parameters $\alpha = 0,71008$ and $\beta = 4,4001$, what is shown in Figure 5c. Figure 5d details the same distribution. Most of 75% of the classes have no public field. Classes with 1 to 10 public fiels is quite rare, and the probability that a class has more than 10 public fields tends to zero. This is another sign



(a) (b)

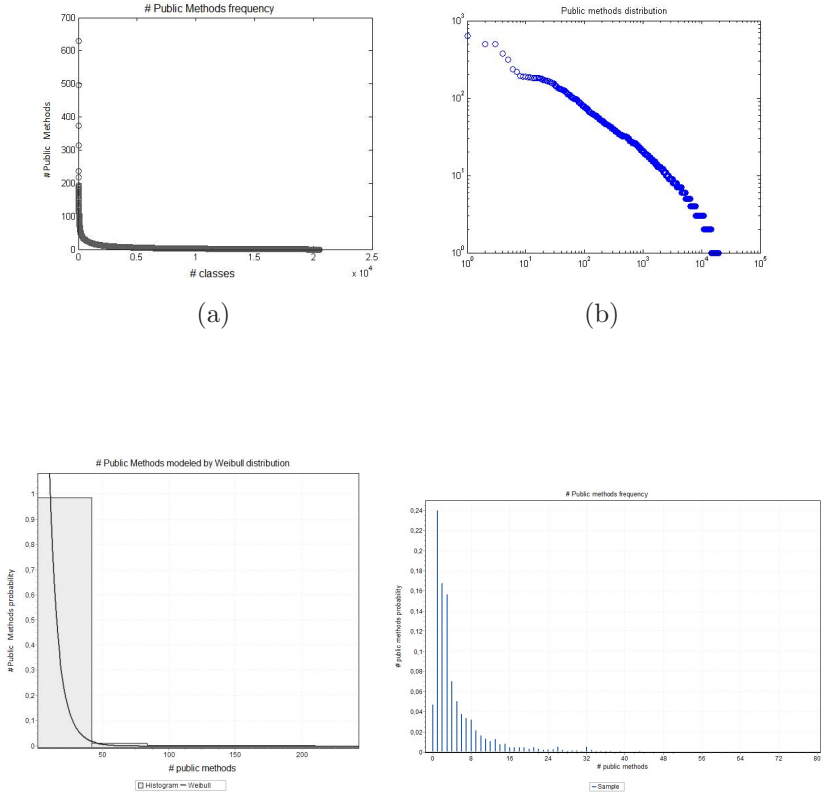
(c) (d)

Figure 5: Public Fields - (a) frequency, (b) frequency in log-log scale, (c) fitting to Weibull distribution and (d) frequency detailed.

of good quality of open source software, because indicates that this kind of software, in general, is not prone to strong module coupling.

4.1.6. Public Methods

The frequency of number of public methods is shown in Figure 6a and, in a log-log scale, in Figure 6b. These plots show that number of public methods follows power law. This metric can be modeled by Weibull distribution, with parameters $\alpha = 0,85938$ and $\beta = 5,6558$, as shown in Figure 6c. The frequency of number of public methods is detailed in Figure 6d. Graphical analysis of data shows that there is a low portion of classes with a great



(a) (b)

(c) (d)

Figure 6: Public Methods - (a) frequency, (b) frequency in log-log scale, (c) fitting to Weibull distribution and (d) frequency detailed.

number of public methods and most classes have few public methods. Most classes have 0 to 10 public methods, classes with 10 to 40 public methods are rare and the probability that a class has more than 40 public methods is quite low. By these findings, it could be concluded that, in most of cases, classes have narrow interface.

Table 3: General Thresholds for OO software metrics

Factor	*Level*	*Metric*	*Reference Values*
Connectivity	System	COF	Good: up to 0,02 Regular: 0,02 to 0,14 Bad: greater than 0,14
	Class	# Afferent couplings	Good: up to 1 - Regular: 2 to 20 - Bad: greater than 20
Information hiding	Class	# Public fields	Good: 0 - Regular: 1 to 8 - Bad: greater than 8
Interface size	Class	# Public methods	Good: 0 to 10 - Regular: 11 to 40 - Bad: greater than 40
Inheritance	Class	DIT	Typical value: 2
Cohesion	Class	LCOM	Good : 0 - Regular: 1 to 20 - Bad: greater than 20

5. Software Metrics Thresholds

A large number of object-oriented open source software was evaluated by means of six software metrics in this study. Findings in this study suggest that open source software has good quality in general: classes are low connected each other, have high cohesion, few public methods and fields, and also short inheritance tree. Considering this, the characteristics of this type of software may be used as target to software development, and measures of their software metrics can be taken as reference. We, then, propose to use the most commonly software metrics measures founded in practice as thresholds. From the achieved results, it is possible to identify three ranges of reference values to the metrics: *good*, which refers to the most common values of the measures of the metric in open source software, *regular*, which is an intermediate range that refers to values with low frequency but not irrevelant, and *bad*, that refers to values with quite rare occurrences. For instance, LCOM, which frequency is shown in Figure 3d, is 0 for more than 44% of the classes, values between 1 and 20 occur in a very low frequency, and values greater than 20 are rare. Therefore, we derive the LCOM threshold: 0 (good cohesion), 1 to 20 (regular cohesion) and greater than 20 (bad cohesion). The same analysis is performed for the other metrics. References values suggested for COF, LCOM, DIT, afferent couplings, number of public methods and number of public fields are sumarized in Table 3. We also carried out a similar analysis by application domains, type and size of the software systems. The results of this analysis are described in Sections 5.1, 5.2 and 5.3, respectively.

5.1. Thresholds for OO Software Metrics by Application Domains

The software metrics studied in this work were analised for each application domain of the software systems listed in Table 1. We founded that a software metric, in a specific application domain, can be modeled for the same probability distribution that fits the metric in the entire data set. Using the same analysis performed to the entire data set, we derived the reference values by application domains reported in Table 4. There is a slight difference between the results of the application domains, and these results do not disagree with that founded to the entire data set. This is true even for the *Financial* application that has only one software system analised in this work. So, we believe that the general result, listed in Table 3, can be used to software systems in general. However, it is necessary a more detailed analysis by application domains, since the sample of each application domain is small.

Table 4: Thresholds for OO software metrics by application domains

Application Domain	*Afferent Coupling (good/regular/bad)*	*# Public fields (good/regular/bad)*	*# Public methods (good/regular/bad)*	*DIT (typical value)*	*LCOM (good/regular/bad)*
Clustering	0 - 1 / 1 - 20 / >20	0 / 1 - 7 / >7	0 - 20 / 20 - 45 / >45	2	0 / 1 - 20 / >20
Database	0 - 1 / 2 - 20 / >20	0 / 1 - 8/ >8	0 - 20 / 21 - 50 / >50	2	0 / 1 - 20/ >20
Desktop	0 - 1 / 2 - 20 / >20	0 / 1 - 8/ >8	0 - 20 / 21 - 55/ >55	2	0 / 1 - 15 / >15
Development	0 - 1 / 2 - 25 / >25	0 / 1 - 8 / >8	0 - 20 / 21 - 35 / >35	2	0 / 1-25 / >25
Enterprise	0 - 1 / 2 - 22 / >22	0 / 1 - 11 / >11	0 - 15 / 16 - 35 / >35	1	0 / 1 - 35 / >35
Financial	0 - 1 / 2 - 16 / >16	0 / 1 - 4 / >4	0 - 13 / 14 - 32 / >32	2	0 / 1 - 25 / >25
Games	0 - 1 / 2 - 22 / >22	0 / 1 - 9 / >9	0 - 20 / 21 - 32 / >32	1	0 / 1-35 / >35
Hardware	0 - 1 / 2 - 11 / >11	0 - 1 / 2 - 6 / >6	0 - 20 / 21 - 36 />36	2	0 / 1 - 80 / >80
Multimedia	0 - 1 / 2 - 20 / >20	0 / 1 - 9 / >9	0 - 35 / 36 - 60 / >60	2	0 / 1 - 60 / >60
Networking	0 - 1 / 2 - 20/ >20	0 / 1 - 5 / >5	0 - 15 / 16 - 40 / >40	2	0 / 1-40 / >40
Security	0 - 1 / 2 - 16 / >16	0 / 1 - 8/ >8	0 - 25 / 26-50 / >50	1	0 / 1 - 45 / >45

5.2. Thresholds for OO Software Metrics by Software Types

Measures were analysed for three types of software: tool, framework and library. Results reveal those metrics have similar behaviour to that detected in the entire data set analysis, regardless the type of software, what confirm

Table 5: Thresholds for OO software metrics by software types

Application Domain	*Afferent Coupling (good/regular/bad)*	*# Public fields (good/regular/bad)*	*# Public methods (good/regular/bad)*	*DIT (typical value)*	*LCOM (good/regular/bad)*
Tool	0 - 1 / 2 - 20 / >20	0 / 1 - 8 / >8	0 - 20 / 21 - 50 / >50	2	0 / 1-35 / >35
Framework	0 - 1 / 2 - 20 / >20	0 / 1 - 10 / >10	0 - 25 / 26-50 / >50	2	0 / 1 - 40 / >40
Library	0 - 1 / 2 - 25 / >25	0 / 1 - 8 / >8	0 - 25 / 26 - 40 / >40	2	0 / 1 - 30 / >30

the hypothesis that there is a single distribution probability that model values of measures of a metric, regardless the type of software.

- Public Fields: in three cases, more than 80% of classes have no public fields, the frequency of classes having 1 to 8 publics fields is very low, and frequency of classes having more than 8 public fields is near to zero.
- Public Methods: results of this metric are also similar in frameworks, libraries and tools. There is a slight difference in tools, whose distribution curve is a little more left concentrated, what indicates that tools have less public methods than frameworks and libraries. This makes sense because both are service providers, while tools are not.
- LCOM: this metric has a very similar distribution in the three cases. As the same result found to the entire data set, 50% of classes have LCOM equal to zero.
- DIT: this metric can be modeled by Poisson distribution in the three cases. There is a little diference in mean values: 1,68 in frameworks, 1,74 in tools and 1,96 in libraries.
- Afferent Couplings: in frameworks, libraries and tools, 20% of classes have no afferent coupling, most of them have 1, frequency of classes with 1 to 20 afferent couplings is low, and classes with more than 20 afferent couplings are rare.

Using the same analysis performed to the entire data set, we derived the reference values by type of software system (tool, framework and library) reported in Table 5.

Table 6: Thresholds for OO software metrics by software size

Application Domain	*Afferent Coupling (good/regular/bad)*	*# Public fields (good/regular/bad)*	*# Public methods (good/regular/bad)*	*DIT (typical value)*	*LCOM (good/regular/bad)*
⩽100	0 - 1 / 2 - 20 / >20	0 / 1 - 10 / >10	0 - 20 / 21 - 30 / >30	2	0 / 1 - 25 / >25
101 - 1000	0 - 1 / 2 - 20 / >20	0 / 1 - 8 / >8	0 - 25 / 6 - 50 / >50	2	0 / 1 - 20 />20
>1000	0 - 1 / 2 - 15 / >15	0 / 1 - 5 / >5	0 - 30 / 30 - 60 / >60	2	0 / 1 - 20/ >20

5.3. Thresholds for OO Software Metrics by Software Size

We grouped the software systems analised in this work in three sets according their size: up to 100 classes, 101 to 1000 classes and more than 1000 classes. The software metrics studied in this work were analised for each set. We founded that a software metric can be modeled by the same probability distribution that fits it in the entire data set, regardless the software system size. Using the same analysis performed to the entire data set, we derived the reference values by software system size reported in Table 6. There is a slight difference between the results founded for the sets. An interesting difference is about the number of public fiels: the higher the software system size, the lower the number of public fields. This makes sense because public fields generate strong coupling between modules in a software system and the larger the software system, the larger the coupling impact on it. Hence public fields tends to be avoided in large software systems. The results of this analysis do not disagree with that founded to the entire data set. Therefore, we believe that the general result, listed in Table 3, can be used to software systems in general, regardless the software system size.

6. Conclusion

The identification of reference values for software metrics is a question in software engineering whose solution can make the use of software metrics effective in the industry. This work presents a study carried out on a large sample of object-oriented open source software and suggest thresholds for a set of object-oriented software metrics. A set of 40 programs developed in Java, including tools, libraries and frameworks, from 11 application domains, was analysed, in a total of more than 26,000 classes. Six software metrics

were used in the study: COF, LCOM, DIT, afferent couplings, number of public methods and number of public fields. The study concluded that those metrics, except DIT, can be modeled by a heavy-tail distribution. This means that, for most metrics, there is a low number of occurrences of high values and a far higher number of occurences of low values. DIT can be modeled by Poisson distribution, with 2 as mean value. This observed characteristic of open source software reveals that this type of software stresses high quality: classes are low connected each other, have high cohesion, few public methods and fields, and also short inheritance tree. This important insight about open source software leads to consider their measures of metrics as reference.

We derived general object-oriented software metrics thresholds, based on the most commonly measures values founded in practice. We also derived thresholds by application domains, software system size and type (tool, library and framework). As the results of this analysis do not disagree with the general object-oriented software thresholds, we believe that this result can be used as reference for objetc-oriented software in general. Nevertheless, it is necessary a more detailed analysis about this assumption because the sample used for each application domain, size and type set is small.

The approach used in the study is suggested to be used in future works to find reference values of other software metrics. The following future works are identified: to extend the study to other programming languages in order to investigate if there are different reference values depending on the programming language; to evaluate the proposed reference values table in a proprietary software development; to extend the study to other software metrics.

This work was sponsored by FAPEMIG, as part of the project CONNECTA Process: CEX APQ-3999-5.01/07. Fapemig.

References

Abreu, F. B., Carapuça, R., October 1994. Object-oriented software engineering: Measuring and controlling the development process. In: Proceedings of 4th Int. Conf. of Software Quality. McLean, VA, USA.

Baxter, G., Frean, M., Noble, J., Rickerby, M., Smith, H., Visser, M., Melton, H., Tempero, E., 2006. Undertanding the shape of java software. In: OOPSLA'06. Oregon, Portland, USA.

Briand, L. C., Daly, J. W., Wüst, J., 1999. A unified framework for cohesion measurement in object-oriented systems. IEEE Transactions on Software Engineering 25, 91–121.

Chidamber, S. R., Kemerer, C. F., 1994. A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 476–493.

Daly, J., Brooks, A., Miller, J., Roper, M., Wood, M., 1996. An empirical study evaluating depth of inheritance on the maintainability of object-oriented software. Empirical Software Engineering 1, 109–132.

Fenton, N. E., Neil, M., 2000. Software metrics: roadmap. In: ICSE '00: Proceedings of the Conference on The Future of Software Engineering. ACM, New York, NY, USA, pp. 357–370.

Ferreira, K. A. M., Bigonha, M. A. S., Bigonha, R. S., 2008. Reestruturação de software dirigida por conectividade para redução de custo de manutenção. Revista de Informática Teórica e Aplicada 15 (2), 155–179.

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

Kitchenham, B., 2009. What's up with software metrics? - a preliminary mapping study. The Journal of Systems and Software 83, 37–51.

Lanza, M., Marinescu, R., 2006. Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. Springer-Verlag, Germany.

Louridas, P., Spinellis, D., Vlachos, V., Setembro 2008. Power laws in software. ACM Transactions on Software Engineering and Methodology 18 (1).

Mathwave, 2010. EasyFit. http://www.mathwave.com/products/easyfit.html, acesso em Fevereiro de 2010.

Meyer, B., 1997. Object-oriented software construction, 2nd Edition. Prentice Hall International Series, Estados Unidos.

Newman, M. E. J., 2003. The structure and function of complex networks. In: SIAM Reviews. Vol. 45. pp. 167–256.

Potantin, A., Noble, J., Frean, M., Biddle, R., Maio 2005. Scale-free geometry in oo programs. Communications of the ACM 48 (5), 99–103.

Puppin, D., Silvestrini, F., 2006. The social network of java classes. In: SAC'06. Dijon, França, pp. 1409–1413.

Samoladas, I., Stamelos, I., Angelis, L., Oikonomou, A., Outubro 2004. Open source software development should strive for even greater code maintainability. Communications of the ACM 47 (10), 83–87.

Sommerville, I., 2000. Software Engineering, 6th Edition. Addison-Wesley.

Tempero, E., 2008. On measuring java software. In: ACSC2008 - Conferences in Research and Practice in Information Technology (CRPIT). Vol. 74. Wollongong, Australia.

Wheeldon, R., Counsell, S., Setembro 2003. Power law distributions in class relationships. In: Proceedings of 3rd International Workshop on Source Code Analysis and Manipulation (SCAM).

Xenos, M., Stavrinoudis, D., Zikouli, K., Christodoulakis, D., 2000. Object-oriented metrics - a survey. In: FESMA 2000. Madrid, Spain.