Reference Values for Six Object-oriented Software Metrics

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

DCC – ICEx – Universidade Federal de Minas Gerais (UFMG)
Av. Antônio Carlos, 6627 - Pampulha - CEP: 31270-010 - Belo Horizonte - Brazil
{kecia,mariza,bigonha,lfmendes,heitorca}@dcc.ufmg.br,
WWW home page: www.dcc.ufmg.br

Abstract. Software metrics allow measurement, evaluation, control and improvement of software products and processes. 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 a study on the structure of a large collection of open source software developed in Java, from different sizes and 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 insigths 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 in other software metrics in order to find the reference values.

1 Introduction

Software metrics allow measurement, evaluation, control and improvement of software products and processes. A great deal of researches have been made in order to define and evaluate these software metrics [1–4, 14]. Despite their importance and the large number of proposed metrics, they have not been applied widely in the industry yet. Tempero [12] believes that one reason for this is that for most metrics typical values for measurements are not known. If software engineers does 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.

Maintainability is a software quality factor that can be improved by internal software factors, such as low coupling between modules and high cohesion inside

modules. Open source software development should emphasize maintainability because software can be freely used and modified. There is a great amount of software of this type. Sourceforge (www.sourceforge.net), for instance, has more than 176,000 avalialable software. According to Samoladas et al. [11], open source software 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. Since maintainability is one of the most important software quality factors and is a condition to open source software success, the characterizantion os this type of software can bring important insights about its quality. And moreover, measures of metrics collected from open source software can be used to define an initial reference for values of these metrics. The objective of this research is the characterization of open source software by means of six software metrics. Based on the results of the study, reference values of these metrics are identified for these metrics.

The paper is organised as follows. Section 2 discusses the most relevant related works. 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 brings the conclusions and suggested future works.

2 Related Works

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 [2, 7, 9, 10, 15]. 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, $P(X = x) \propto cx^{-k}$.

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 nets, such as in-degree in Internet nodes [8].

Some researches have identified power laws in graphs that represent relationship between software classes and objects in an object oriented system. Potanin et al. [9] analysed 60 graphs of 35 software and concluded that the geometry of relationship between objetcs, 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, in a scale free graph this is not true. Wheeldon and Counsell [15] identified power laws in classes relationship in Java programs. Their study envolved three well-known software: 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. [7] 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 was analysed, such as J2SE SDK, Eclipse, OpenOffice and Ruby. The study concludes that, regardless programming paradigm, in and out-degree are governed by power law.

A large number of metrics have been proposed [1, 3, 14]. Tempero [12] points out that knowing the typical values of software metrics is necessary in order to interpret the measurements. Some researches have been made in order to study structure of software, but the typical values of most software metrics are not known yet. The study of Baxter et al. [2] 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 the study of Baxter et al. is from 56 open source software, varying size and 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. [7], that founded power law in out-degree of classes. Findings of Baxter et al. [2] and Louridas et al. [7] are very 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 analised metrics of important quality factors, such as module coupling and cohesion.

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) [3], COF (coupling factor) [1], afferent couplings, number of public methods and number of public fields. 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 [11], 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. Program codes are from 11 application domains and three types: tool, library and framework. In total, 26,202 classes were analised. Software names and their domains are described in Table 1. A tool, called *Connecta* [5], was used to collect measures. *Connecta* collects measures of object oriented software metrics from

bytecodes of Java programs. Because of this, a criteria to choose software to be analysed in this study it was the bytecode availability.

Three types of analysis are made on the collected measures: to the entire data set, by application domain and by type of software. The hypothesis investigated is: there is a single distribution probability 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[3], and the MOOD set metrics, by Abreu and Carapuça[1]. In this study, the following metrics are used:

COF (Coupling Factor): this metric is calculated to the system. It is based on the concept of *client-server* relationship between classes. Considering 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. So, software can be modeled as a directed graph. In this study, when A is a subclass of B, it is also considered there is a connection from A to B. 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. Higher COF value, higher the connectivity of the system and low its maintainability [1].

LCOM (Lack of Cohesion in Methods): this metric is calculated for a class. It considers the concept of similarity of methods. Two methods are similar if they use at least one common field of their class. LCOM is given by the number of pairs of similar methods minus the number of pairs of methods without similarity. Higher LCOM value, lower the class cohesion [3].

DIT (Depth of Inheritance Tree): this metric is calculated for a class. It is the maximum distance of a class from the root class in the inheritance tree. It is considered an indicator of the design complexity, because higher the depth in the inheritance tree, more complex the behavior of the class, because more classes are involved [3].

Afferent couplings: this metric is calculated for a class. It is given by the total number of classes that use services of the class, what involves method invocations, field uses and subclasses. Classes having high number of afferent couplings play important role in the system, because errors or modifications on them can widely impact the other ones.

Table 1. software used in the study, their application domain, type, size and COF metric

Domain	*Software*	*Type*	*#Classes*	*#Conexões*	*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	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
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	29	16	0,02
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

Number of public methods: this metric is calculated for a class. This metric indicates the interface size of the class.

Number of public fields: this metric is calculated for a class. Using public fields can generate stronger types of coupling between classes.

3.2 Data Fitting

A tool, called EasyFit, 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.

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

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

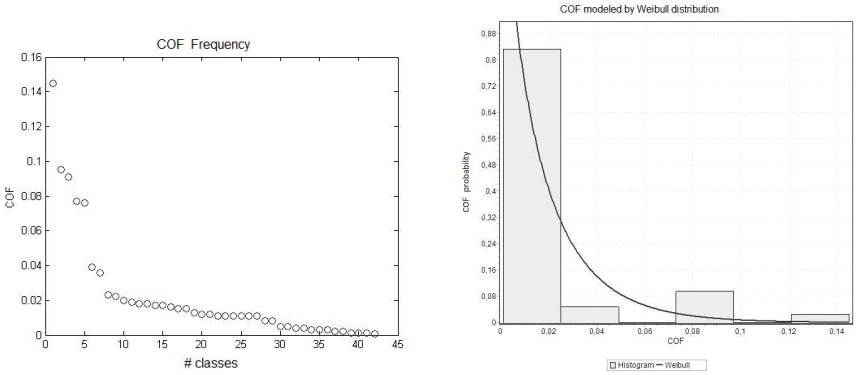
3.3 Data Analysis

For each metric, data were collected and a scatter plot was generated in order to exhibit the frequency of measures. 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 Normal and Poisson distributions, 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 are thought to be well constructed in order to facilitate their maintainance and use, even without documentation or technical support, characteristics of this type of software can be taken as a model, and the measures of their metrics can be taken as reference values. Section 4 describes the findings of this study.

4 Results

This section describes the findings of this study. First, results of COF are described. Results of the other metrics that evaluate classes are described next.

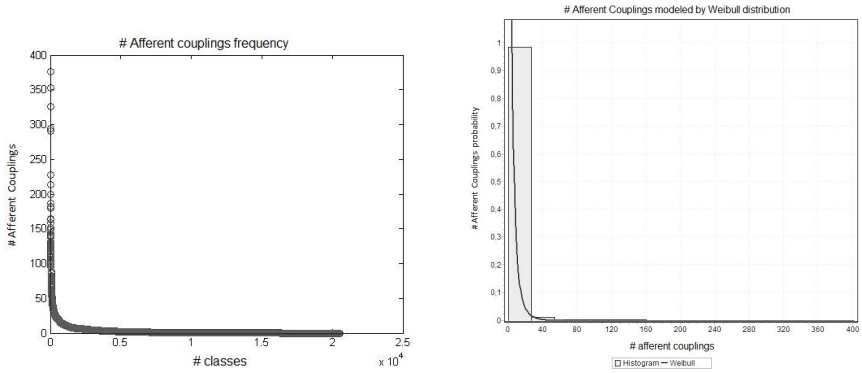


(a) (b)

Fig. 1. COF distribution - fitting to Weibull distribution.

4.1 COF

COF scatter plot, in Figure 1, 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 1 shows fitting data with Weibull distribution. Based on graphic analysis, more than 80% of software 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 contributes to high maintainability.



(a) (b)

Fig. 2. Afferent Couplings distribution - fitting to Weibull distribution.

4.2 Metrics for Classes

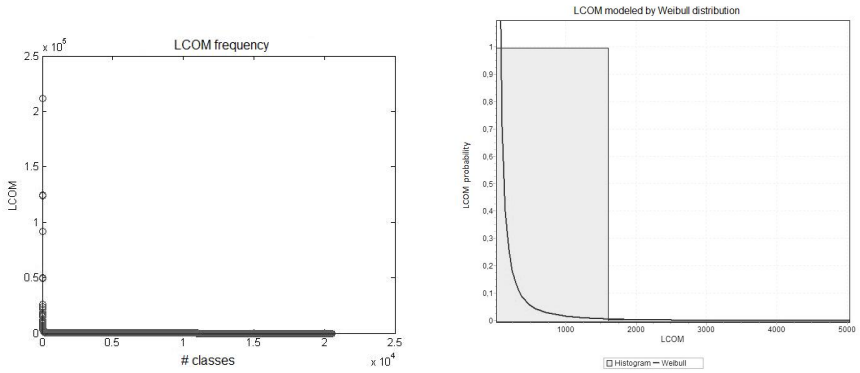
Afferent Couplings

Scatter plot for afferent couplings, shown in Figure 2, suggests a heavy-tail distribution. There is a small number of classes with high number of afferent couplings and a far higher number of classes with few afferent couplings. Values of this metric can be modeled by Weibull distribution, with parameters $\alpha = 0,78986$ and $\beta = 3,2228$. 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.

LCOM

LCOM also is fitted by a heavy tail distribution. Figure 3 shows scatter plot of the data set. Values of LCOM can be modeled by Weibull distribution, with parameters $\alpha = 0,23802$ and $\beta = 1,465$. 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.



(a) (b)

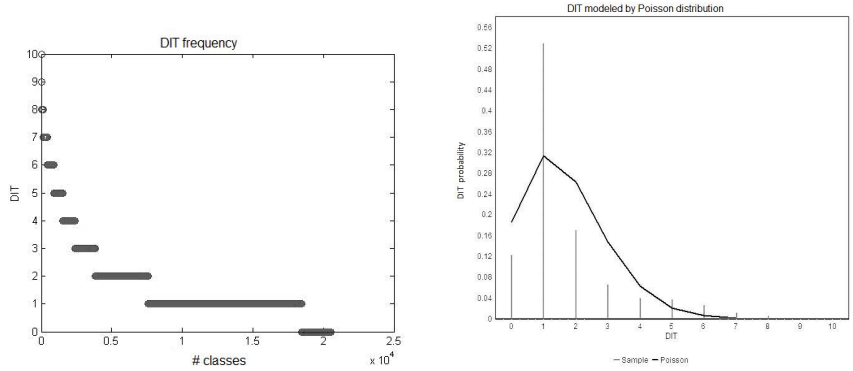
Fig. 3. LCOM distribution - fitting to Weibull distribution.

DIT

Scatter plot in Figure 4 shows distribution of DIT values and their fitting to Poisson distribution, 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 has not very deep inheritance tree, what also contributes to software maintainability by decreseasing software complexity.

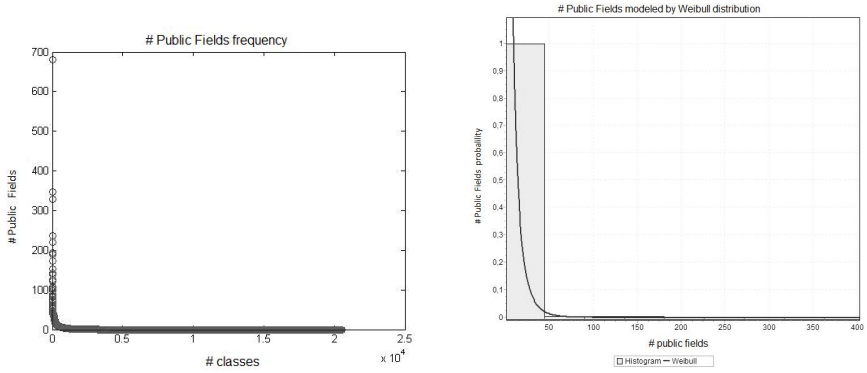
Public Fields

Scatter plot of number of public fields, shown in Figure 5, reveals that there is a low number of classes with a great number of public fields and, in most



(a) (b)

Fig. 4. DIT distribution - fitting to Poisson distribution.



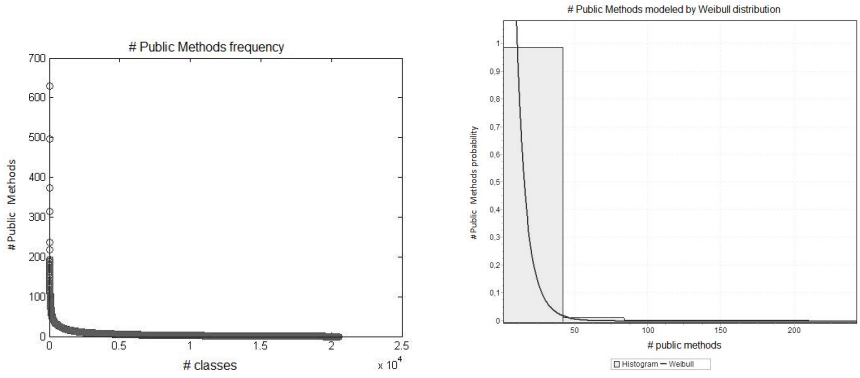
(a) (b)

Fig. 5. Public Fields distribution - fitting to Weibull distribution.

cases, this number is zero. This metric can be modeled by Weibull distribution with parameters $\alpha = 0,71008$ and $\beta = 4,4001$, what is showed in Figure 5. Most of 75% of the classes have no public field. Classes with 1 to 8 public fiels is quite rare, and the probability that a class has more than 8 public fields tends to zero. This is another sign of high quality of open source software, because it demonstrates that development of these software strictly follow information hiding.

Public Methods

Frequency of number of public methods is shown in Figure 6. This metric can be modeled by Weibull distribution, with parameters $\alpha = 0,85938$ and $\beta = 5,6558$. Graphical analysis of data shows that there is a low portion of classes with a great 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 this findings, it could be concluded that, in most of cases, classes have short interface. This could be related to the fact that most classes in open source software have high cohesion and, then, they tend to provide less services.



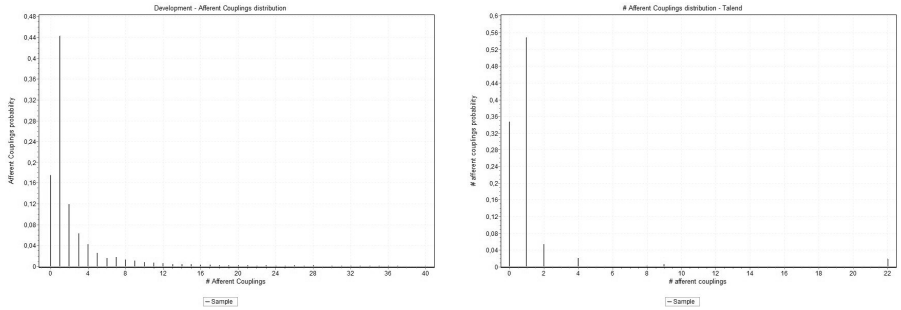
(a) (b)

Fig. 6. Public Methods distribution - fitting to Weibull distribution.

4.3 Data Fitting of Metrics in Application Domains and a Particular Software

Software metrics in a specific application domain can be modeled for the same probability distribution that fits the metric in the entire data set. It also

happens to measures in a particular software. Figure 7 ilustrates distribution of afferent couplings in *development* domain and in Talend (sourceforge.net). This indicates that observed characteristics in open source software, in general, can be applied to a particular software, regardless its application domain.



(a) (b)

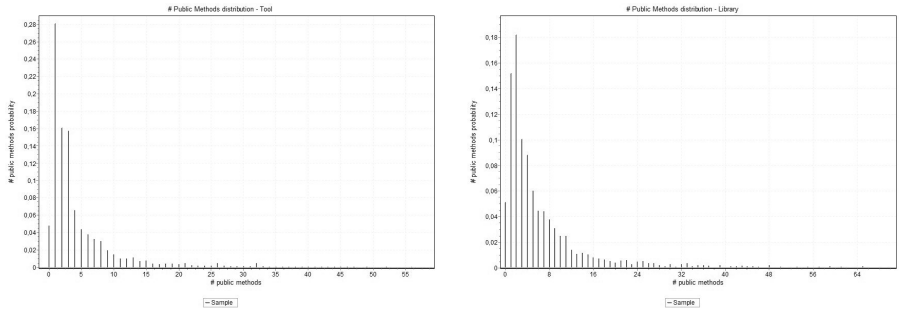
Fig. 7. Afferent Couplings - *Development* and Talend.

4.4 Data Fitting of Metrics in Different Types of Software

Measures were analysed for three types of software: tool, framework and library. Surprisingly results reveal those metrics have similar behaviour to that detected in the entire data set analysis, regardless the type of software, what confirm the hypothesis that there is a single distribution probability that model values of measures of a metric, regardless the application domain and 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 provider, while tools are not. Figure 8 shows distribution of measures of this metric in tools and libraries.

– LCOM: this metric have a very similar distribution values 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. Frameworks and tools have close results, what can be explained by the fact that frameworks represent applications characteristics, so these characteristics are replicated in applications like tools.
– 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.



(a) (b)

Fig. 8. Public Methods distribution in tools and libraries

4.5 Reference Values

A large number of object oriented open source software was evaluated by means of six software metrics in this study. Findings in this study confirm the intuitive notion that open source software has high quality. Considering this, the characteristics of this type of software can be used as target to software development, and measures of their software metrics can be taken as reference. 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. References values suggested for COF, LCOM, DIT, afferent couplings, number of public methods and number of public fields are sumarized in Table 2.

Table 2. Reference values for OO software metrics

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

5 Conclusion

This work presents a study carried out on a large sample of object oriented open source software. A set of 40 software developed in Java, including tools, libraries and frameworks, from 11 application domais, 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.

This is one of the first studies towards the identification of reference values for software metrics, an open question in software engineering whose solution can make the use of software metrics effective in the industry. 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 suggested: 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 - Conectividade em Módulos. Process: CEX APQ-3999-5.01/07. Fapemig - Edital Universal.

References

1. Abreu, Fernando Brito; Carapuça, Rogério. (1994). *Object-Oriented Software Engineering: Measuring and Controlling the Development Process.* In: Proceedings of 4th Int. Conf. of Software Quality, McLean, VA, USA, 3-5 October 1994.
2. 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, 22-26 October 2006.
3. Chidamber, Shyam R.; Kemerer, C.F. (1994) *A Metrics Suite for Object Oriented Design.* IEEE Transactions on Software Engineering, pp. 476-493, 1994.
4. Fenton, Norman; NEIL, Martin. (2000) *Software Metrics: Roadmap.* In: Proceedings of the Conference on the Future of Software Engineering, Maio de 2000.
5. Ferreira, Kecia A. M.; Bigonha, Mariza. A. S.; Bigonha, Roberto. S.(2008) *Reestruturação de Software Dirigida por Conectividade para Redução de Custo de Manutenção.* Revista de Informática Teórica e Aplicada. Vol. 15. No 2. Rio Grande do Sul, Brazil: 2008. pp: 155-179.
6. Ferreira, Kecia A. M.; Bigonha, Mariza. A. S.; Bigonha, Roberto. S.; Almeida, H. C.; Mendes, L. F. O.(2009) *Valores Referencia de Métricas de Software Orientado por Objetos.* In Proceedings of SBES (Simpósio Brasileiro de Engenharia de Software). Fortaleza, Brazil: 2009.
7. Louridas, P.; Spinellis, D.; Vlachos, V. (2008) *Power Laws in Software.* ACM Transactions on Software Engineering and Methodology, Vol. 18, No 1, Article 2. Setembro de 2008.
8. Newman, M. E. J.(2003) *The structure and function of complex networks.* SIAM Reviews, Vol. 45. No 2, pp: 167-256.
9. Potantin, A.; Noble, J.; Frean, M.; Biddle, R.. (2005) *Scale-Free Geometry in OO Programs.* Communications of the ACM. May 2005. Vol. 48. N° 5. pp. 99-103.
10. Puppin, D.; Silvestrini, F.. (2006) *The Social Network of Java Classes.* SAC'06. Abril de 2006. Dijon, França. pp. 1409-1413.
11. Samoladas, I; Stamelos, I; Angelis, L; Oikonomou, A. (2004) *Open Source Software Development Should Strive for Even Greater Code Maintainability.* Communications of the ACM. October 2004/Vol. 47. no 10.pp. 83-87.
12. Tempero, Ewan. (2008) *On Measuring Java Software.* In: ACSC2008. Wollongong, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 74. 2008.
13. Valverde, S.; Ferrer-Cancho, R.; Sole, R.. (2002) *Scale-free networks from optimal design.* Europhysics Letters 60, 4. Novembro de 2002. pp 512-517.
14. Xenos, M.; Stavrinoudis, D.; Zikouli, K.; Christodoulakis, D. (2000) *Object-Oriented Metrics - A Survey.* Proceedings of the FESMA 2000, Madrid, Spain, 2000.
15. Wheeldon, R.; Counsell, S.. (2003) *Power law distributions in class relationships.* In: Proceedings of 3rd International Workshop on Source Code Analysis and Manipulation (SCAM), Amsterdam, Setembro de 2003.