Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2010, Article ID 729826, 21 pages
doi:10.1155/2010/729826

Research Article

An Empirical Study of Social Networks Metrics in Object-Oriented Software

Giulio Concas, Michele Marchesi, Alessandro Murgia, and Roberto Tonelli

Department of Electrical and Electronic Engineering, University of Cagliari, 09123 Cagliari, Italy

Correspondence should be addressed to Roberto Tonelli, roberto.tonelli@dsf.unica.it

Received 14 April 2010; Accepted 13 August 2010

Academic Editor: Ewan Tempero

We study the application to object-oriented software of new metrics, derived from Social Network Analysis. Social Networks metrics, as for instance, the EGO metrics, allow to identify the role of each single node in the information flow through the network, being related to software modules and their dependencies. These metrics are compared with other traditional software metrics, like the Chidamber-Kemerer suite, and software graph metrics. We examine the empirical distributions of all the metrics, bugs included, across the software modules of several releases of two large Java systems, Eclipse and Netbeans. We provide analytical distribution functions suitable for describing and studying the observed distributions. We study also correlations among metrics and bugs. We found that the empirical distributions systematically show fat-tails for all the metrics. Moreover, the various metric distributions look very similar and consistent across all system releases and are also very similar in both the studied systems. These features appear to be typical properties of these software metrics.

1. Introduction

Measuring software to get information about its properties and quality is one of the main issues in modern software engineering. Limiting ourselves to object-oriented (OO) software, one of the first works dealing with this problem is the one by Chidamber and Kemerer (CK), who introduced the popular CK metrics suite for OO software systems [1]. Other OO metrics have been also proposed, like MOOD [2] and the Lorenz and Kidd metric suite [3], but the CK suite remains by far the most widely used. In fact, different empirical studies showed significant correlations between some of CK metrics and bug-proneness [4–7].

Modern software systems are made of many elementary units (software modules) interconnected in order to cooperate to perform specific tasks. In particular, in OO systems the units are the classes, which are in turn interconnected with each other by relationships like inheritance and dependency. Recently, it has been shown how these software systems may be analyzed using complex network theory [8–10]. In software networks, the classes are the nodes and the relationships among classes are the edges. This property opens the perspective to analyze software networks using metrics taken from other disciplines, like Social Network Analysis (SNA) [11]. These SNA metrics can be used together with more traditional product metrics, like class LOCS, number of Bugs, or the CK suite, to gain a deeper insight into the properties of software systems. Recent studies showed the importance of SNA metrics in measuring the interactions among software modules [12], and in particular how centrality measures are useful to identify software hubs, which show higher defect-proneness.

Considering software systems as graphs is not a new approach, and different authors have already investigated some of their properties, like the distribution of Fan-in or Fan-out of network nodes [8, 13], finding features characteristic of complex networks, like for instance, the presence of power-laws in the tail of the distributions of these metrics.

Only recently, SNA has been applied to the study of software systems. Zimmermann and Nagappan used SNA metrics to investigate a network of binary dependencies [12]. With regard to the study of OO software systems, only Tosun et al., to the authors' knowledge, applied SNA metrics to OO source code to assess defect prediction performance of these metrics [14]. In particular, there are no studies investigating the relationships and the correlations among SNA metrics, traditional metrics, and Bugs metrics, and the corresponding statistical distributions. It must be noted that, when the measures are distributed according to power-laws, or other leptokurtotic distributions, traditional quantities like average or standard deviation may lose their meaning, and may not be characterizing measures anymore [15]. Knowledge of the overall statistical distribution is needed for characterizing the system properties. In particular, they are needed in order to obtain estimates of the metrics values for the future software releases.

In this paper, we study a set of releases of two large Open Source OO systems, Eclipse [16] and NetBeans [17], from the software network perspective, and compute the observed complementary cumulative distribution functions (CCDF) [15] of several metrics including SNA metrics. We study these systems because both the source code of several versions and complete data about Bugs and Issues of their software modules are available.

We study the relationships between these metrics and software fault-proneness—measured as the number of Bugs affecting software modules—and between them and more traditional software metrics. We also study the possibility of estimating the metric features for the future releases. For all the observed distributions, we performed best fits, finding analytical distributions able to model the system.

The systems analyzed are written in Java. All their classes are contained in Java source files, called Compilation Units (CU). A CU generally contains just one class, but less frequently it may contain two or more classes. We extracted the Bugs affecting files merging information found in bug-tracking repositories, specifically Bugzilla [18] and Issuezilla [19], with information taken from source code repositories, namely, Concurrent Versioning System (CVS) [20]. The information about Bugs and software changes (commit logs) is reported at CU level, and not at class level. Therefore, we extended the concept of software graph to CU level, building a graph in which nodes are Compilation Units and edges are the relationships between these CU's, extracted from the classes belonging to each CU. We used this graph for computing all the metrics analyzed as well as for computing the Bug distributions.

We found that most of the studied metrics are distributed according to the Yule-Simon distribution [21, 22], to a high degree of accuracy, and show a persistent or universal character across all different releases, for both systems analyzed. The high degree of accuracy of the analytical fitting distributions and their persistent character allowed us to estimate metrics values for the subsequent releases.

The paper is organized as follows. In Sections 2 and 3, we present related works and the research questions. In Section 4, we describe the software network obtained considering Compilation Units as nodes and redefine the CK metric suite for this case. Section 5 defines the SNA metrics analyzed. In Section 6, we discuss the process of Bug recovery and how to assign a Bug to the proper CU. In Section 7, we present the analytical distribution functions used to describe the empirical data and provide the main results. Section 8 analyses correlations between metrics and between metrics and Bugs. Section 9 shows how it is possible to use the results of previous sections to provide estimate for the feature of future releases. In Section 10, we discuss our finding and present the conclusions.

2. Related Work

Product metrics, extracted by analyzing static code of software, have been used to build models that relate these metrics to failure-proneness [4–7, 23]. Among these, the CK [24] suite is historically the most adopted and validated to analyze bug-proneness of software systems [4–7]. CK suite was adopted by practitioners [4] and is also incorporated into several industrial software development tools. Based on the study of eight medium-sized systems developed by students, Basili et al. [5] were among the first to find that OO metrics are correlated to defect density. Considering industry data from software developed in C++ and Java, Subramanyam and Krishnan [6] showed that CK metrics are significantly associated with defects. Among others, Gyimóthy et al. [7], studying an Open Source system, validated the usefulness of these metrics for fault-proneness prediction.

CK metrics are intended to measure the degree of coupling and cohesion of classes in OO software contexts. However, the studies using CK metrics do not consider the amount of "information" passing through a given module of the software network. Social Network Analysis (SNA) fills this gap, providing a set of metrics able to extract a new, different kind of information from software projects. Recently, this ability of SNA metrics was successfully employed to study software systems. Zimmermann and Nagappan [12] showed that network measures derived from dependency graphs are able to identify critical binaries of a complex system that are missed by complexity metrics. However, their results are obtained considering only one industrial product (Windows Server 2003). Tosun et al. [14] reproduced the previous work [12] extending the network analysis in order to validate and/or refute its results. They show that network metrics are important indicators of defective modules in large and complex systems. On the other hand, they argue that these metrics do not have significant effects on small scale projects. Both previous studies [12, 14] did not consider mutual relationships among SNA metrics and complexity metrics; therefore, they did not show if SNA metrics carry new information with respect to CK suite. Our work, instead, computes the correlation matrix among SNA metrics and CK metrics, considering also mutual correlations with respect to Issue, Bug, LOC, Fan-out and Fan-in.

3. Research Questions

The Pareto principle (80–20 rule) and the presence of power-laws in the tail of the distributions of many properties of software systems, including Bugs, have already been observed [9, 25, 26]. In [27], a high-order statistic coefficient was proposed to analyze software metrics exhibiting highly skewed statistical distributions, that was efficient in observing changes in software systems and in monitoring the development process.

We investigate if the new proposed SNA metrics possess the same properties and have similar empirical distributions. Moreover, the new metrics might possibly show correlations with Bugs and/or with other metrics and properties. Thus, it is desirable to study these correlations.

We also investigate if there are analytical distribution functions which may be used to describe such empirical distributions and possibly to forecast future properties of the software systems.

Consequently, our research questions are the following.

(i) RQ1: Are there analytical distribution functions describing the empirical data? Have these functions power-law behavior in their tails? What is the significance level of fitting empirical data with these distributions?

(ii) RQ2: Are these distributions similar in all the releases and in different systems, or tend to vary significantly?

(iii) RQ3: Is it possible to use these distributions to estimate the metrics values in subsequent releases?

(iv) RQ4: Are there SNA metrics significantly correlated with software Bugs, and to which extent?

(v) RQ5: Are there SNA metrics significantly correlated to traditional CK metrics, and to which extent?

4. CU Software Networks and CU-CK Metrics

An oriented graph can be associated to an OO software system, whose nodes are classes and interfaces, and whose edges are the relationships between classes, namely, inheritance, composition, and dependence. This approach has already been used in the literature. In [28] complex software networks were analyzed with nodes representing software entities at any level, and links representing syntactical relationships between modules, subprograms, and instructions. In [13] software is seen as a network of interconnected and cooperating components, choosing modules of varying size and functionalities, where the links connecting the modules are given by their dependencies. In [12] nodes are binaries, and edges are dependencies among binary pieces of code. In [29] interclass relationships were examined in three Java systems, and in [30] the same analysis was replicated on the source code of 56 Java applications. Object graphs were analyzed in [10] in order to reveal scale-free geometry of object-oriented programs, where the objects were the nodes and the links among objects were the network edges.

All these studies were devoted to exploit general dependencies among pieces of code in different software modules. With the same aim, in our study we do not distinguish between the various possibilities of software relationships, and with regard to SNA metrics, for simplicity, we do not even consider edges orientation, which would imply the construction of different EGO networks for the different kinds of links. Ours is a static analysis. Furthermore, since our software nodes are CUs, as explained later, many relationships among Java classes lose their original meaning at this granularity level. Our purpose is to focus on the role of the interactions among the software elements.

The number and orientation of edges allow to study the coupling between nodes, that is between classes. In this graph, the in-degree of a class, or Fan-in, is the number of edges directed toward the class. It measures how much this class is used by other classes of the system. The out-degree of a class, or Fan-out, is the number of edges leaving the class. It represents the level of usage the class makes of other classes in the system. Besides Fan-in and Fan-out metrics, we computed also, for each class, four CK metrics which were observed to be significantly correlated with the number of Bugs. They are as follows.

(i) Weighted Methods per Class (WMC). A weighted sum of all the methods defined in a class. We set the weighting factor to one, to simplify our analysis.

(ii) Coupling Between Objects (CBO). The counting of the number of classes which a given class is coupled to.

(iii) Response For a Class (RFC). The sum of the number of methods defined in the class, and the cardinality of the set of methods called by them and belonging to external classes.

(iv) Lack of Cohesion of Methods (LCOM). The difference between the number of noncohesive method pairs and the number of cohesive pairs.

We also computed the lines of code of the class (LOC), excluding blanks and comment lines. This is useful to keep track of the class size, because it is known that a "big" class is more difficult to maintain than a smaller class.

Every class is contained in a Java file, called CU. While most files include just one class, there are files including two or more classes. In Eclipse, about 10% of CUs host more than one class, whereas in Netbeans this percentage is about 30%.

While OO metrics and class graphs are usually referred to classes, Bugs and Issues are typically associated to CUs, because the logs of coding efforts aimed to fix Bugs are associated to changes to the source code, which are made to files (the CUs). Since the number of Bugs is of paramount importance to define software quality, to make Issue tracking consistent with source code we decided to base our analysis on CUs. Consequently, we extended CK metrics from classes to CUs. CUs represent therefore the main element of our study.

We defined a CU graph, whose nodes are the CUs of the system. Two nodes, A and B, are connected with an edge directed from A to B if at least one class inside the CU represented by A has a dependency relationship with one class inside the CU represented by B. Referring to this graph, we can compute In-links and Out-links of a CU-node. We reinterpreted LOC and CK metrics for this CU-graph:

(i) CU LOC is the sum of the LOCS of the classes contained in the CU;

(ii) CU CBO is the number of out-links of each node, excluding those representing inheritance. This definition is consistent with that of CBO metrics for classes;

(iii) CU LCOM and CU WMC are the sum of LCOM and WMC metrics of the classes contained in the CU, respectively;

(iv) CU RFC is the sum of weighted out-links of each node, each out-link being multiplied by the number of specific distinct relationships between classes belonging to the CUs connected to the related edge.

For each, CU we have thus a set of 7 metrics: In-links (Fan-in), Out-links (Fan-out), CU-LOCS, CU-LCOM, CU-WMC, CU-RFC, and CU-CBO. These metrics were computed for CUs of all versions of Eclipse and Netbeans.

5. SNA Metrics

Once the CU software graph is defined, we can compute on this graph the metrics used in Social Network Analysis. We restricted ourselves to the subset of SNA metrics that were found most correlated to software quality [12, 31]. Some of these metrics are the so-called "EGO metrics". For every node in the graph, there exists a subgraph composed by the node itself, called "EGO" (from the Latin word "ego", meaning "I"), and its immediate neighbors. Such subgraph is called the EGO Network associated to the node. The analysis of the EGO-networks gives information about the role of the "EGO" inside the entire network. In particular, EGO-network metrics provide insights on the extent each CU is connected to the entire system, and on the flow of information. In the definition of the EGO network, we considered the graph links as undirected links.

Other SNA metrics we considered, not directly related to the EGO network, are some centrality metrics, determining how important a given node/edge is relative to other nodes/edges in the network. Overall, we consider the following SNA metrics.

(i) Size: size of the EGO-network related to the considered node (i.e., Compilation Unit); it is the number of the nodes of the EGO-network.

(ii) Ties: number of edges of the EGO-network related to the node.

(iii) Brokerage: the number of pairs not directly connected in the EGO network, excluding the EGO node.

(iv) Eff-size: effective size of the EGO network; the number of nodes in the EGO network minus one, minus the average number of ties that each node has to other nodes of the EGO network.

(v) Nweak-comp: normalized Number of Weak Components; the number of disjoint sets of nodes in the EGO network without EGO node and the edges connected to it, divided by Size.

(vi) Reach-Efficiency; the percentage of nodes within two-step distance from a node, divided by Size.

(vii) Closeness; the sum of the lengths of the shortest paths from the node to all other nodes.

(viii) Information Centrality: the harmonic mean of the length of paths starting from all nodes of the network and ending at the node.

(ix) DwReach: the sum of all nodes of the network that can be reached from the node, each weighted by the inverse of its geodesic distance. The weights are thus 1/1, 1/2, 1/3, and so on.

All previous metrics are computed on the CU graph and are among those studied in [12]. It is useful to shortly describe how these SNA metrics may be relevant to software systems, namely, what they try to measure. The first five are strictly EGO metrics and describe the software neighborhood of a CU. Size measures the CU directly connected to a given CU while Ties, measured on such neighborhood, measures how dense are the software connections in this local network. Brokerage measures for how many couples of CU the given node acts like a broker, bridging the information flow among couples. Eff-size measures the redundancy of the connections in the EGO network, reducing the CU Size by an amount proportional to the local average Ties. If the average Ties is high, the local network has in fact redundant channels available for the information flow. The role of the EGO CU in the information exchange is then reduced. It must be noted that the average Ties refers only to the local network, and not to the global network, where, as we will see in the following, the distribution of Ties among all the nodes presents a fat tail. Nweak-comp measures how much the CU is needed to keep connected the other software units. The remainings are not EGO-metrics and are all centrality metrics. They measure if, in the global software network, the CU plays a peripheral rather than a central role.

We analyze the correlations among all of these metrics, as well as with the other metrics and with Bugs. For some metrics, we analyzed the statistical distributions and performed best fits with analytical distribution functions.

6. Issues Extraction

Bug Tracking Systems (BTSs) are commonly used to keep track of Bugs, enhancements, and features—called with the common term "Issues"—of software systems. The open source systems studied, Eclipse and Netbeans, make use of BTS Bugzilla and Issuezilla, respectively.

Each Issue inside a BTS is univocally identified by a positive integer number, the Issue-ID. BTS store, for each tracked Issue, its characteristics, life-cycle, software releases where it appears, and other data. In Bugzilla, a valid Bug is an Issue with a resolution of "fixed", a status of "closed", "resolved", or "verified", and a severity that is not "enhancement", as pointed out in Eaddy et al. [32]. Thus, Bugs are a subset of Issues. For Issuezilla, it is possible to adopt an equivalent definition: a Bug is an Issue with a resolution and status as above, and with type "defect".

Software configuration management systems like CVS (Concurrent Version System) keep track of all maintenance operations on software systems. These operations are recorded inside CVS in an unstructured way; it is not possible, for instance, on query CVS to know which operations were done to fix Bugs, or to introduce a new feature or enhancement. In order to identify Issues (Bugs)

Table 1: Number of CUs of Eclipse for each release.

Release	2.0	2.1	3.0	3.1	3.2	3.3	3.4
Number of CU	6391	7545	10288	11854	14138	15439	17387
Release date	06-2002	03-2003	06-2004	06-2005	06-2006	06-2007	05-2008

Table 2: Number of CUs of Netbeans for each release.

Release	3.2	3.3	3.4	4.0	6.0	6.1
Number of CU	3346	4383	6264	9317	31425	35034
Release date	04-2001	11-2001	08-2002	12-2004	12-2007	04-2008

affecting systems CUs, we had to match data stored in BTS with other data recorded in CVS of Eclipse and Netbeans.

All commit operations are committed to the CVS log messages as single entries. Each entry contains various data—among which the date, the developer who made the changes, a text message referring to the reasons of the commit, and the list of CU's interested by the commit. To obtain a correct mapping between Issue(s) and the related CU(s), the only way is to analyze the CVS log messages, to identify commits associated to maintenance operation where Issues are fixed. If a maintenance operation is done on a CU to address an Issue, we consider the CU as affected by this Issue.

In our approach, we first analyzed the text of commit messages, looking for Issue-IDs. In fact, in commit messages, there may be strings such as "Fixed 141181" or "bug no. 141181", but sometimes only the Issue-ID is reported. Unfortunately, every positive integer number is a potential Issue-ID, but sometimes numbers can refer to maintenance operations not related to Issue-ID resolution, such as branching, data, number of release, and copyright updating.

To avoid wrong mappings between Issue-IDs and CUs, we applied the following strategies.

(i) For each release, a CU can be hit only by Issues which are referred to in the BTS belonging to the same release.

(ii) We did not consider some numeric intervals particularly prone to host false positive Issue-IDs.

The latter condition is not particularly restrictive in our study, because we did not consider the first releases of the studied projects, where Issues with "low" ID appear. All IDs not filtered out are considered Issues and associated to the addition or modification of one ore more CUs, as reported in the commit logs. This method might not completely address the problems in the mapping between bugs and CUs [33]. In any case, we checked manually

- 10% of CU-bug(s) associations (randomly chosen) for each release,

- each CU-bug association for 6 subprojects (3 for Eclipse and 3 for Netbeans) without finding any error. A bias may still remain due to lack of information on CVS [33].

The total number of Issues affecting a CU in each release constitutes the Issue-metric we consider in this study, while the subset of Issues satisfying the conditions as in Eaddy et al. is the Bug-metric [32]. Clearly, not all source modules changed due to a Bug are to be considered "faulty". Some changes can happen to realign a correct piece of code with another piece of code that was modified to fix the Bug. So, what we measure is to what extent a Bug hits one, some, or many CUs, and not whether they were really faulty.

7. Empirical Results Regarding Metric Distributions

We systematically analyzed several main releases of Eclipse and Netbeans projects, namely, releases from 2.0 to 3.4 of Eclipse and releases from 3.2 to 6.1 of Netbeans. For each release, we computed the class graph and the consequent CU graph, and computed all the above quoted metrics at CU level. We analyzed the statistical distributions of the metrics among the systems CU's, which are our graph nodes, as well as the Bugs and Issues distributions. Note that we used CU metrics to be able to study more easily their relationships with Bugs and Issues. However, we verified that the behavior of CU metrics is absolutely similar to the behavior of the corresponding class metrics, for all considered metrics.

Tables 1 and 2 show the number of CUs in the various releases considered of Eclipse and Netbeans, respectively, together with their release date. Both the size and the release date of the considered systems vary considerably. The sizes—in number of CUs—vary of one order of magnitude in Netbeans, and about three times in Eclipse.

In the following figures, we systematically report the experimental CCDF (Complementary Cumulative Distribution Function) in log-log scale, as well as the best-fitting curves in many cases. This is convenient because, if the PDF (probability distribution function) has a power-law in the tail, the log-log plot displays a straight line for the raw data. This is a necessary but by no means a sufficient condition for power-law behavior. Thus we used log-log plots only for convenience of graphical representation, but all our calculations (CDF, CCDF, best fit procedures and the same analytical distribution functions we use) are always in normal scale.

The problems with representing the experimental PDF are that it is sensitive to the binning of the histogram used to calculate the frequencies of occurrence, and that bins with very few elements are very sensitive to statistical noise. This causes a noisy spread of the points in the tail of the distribution, where the most interesting data lie. Furthermore, because of the binning, the information relative to each single data is lost. All these aspects make difficult to verify the power-law behavior in the tail. Thus, we adopted the CCDF representation, which presents various advantages. With this representation, there is no dependence on the binning, nor artificial statistical noise added to the tail of the data. If the PDF exhibits a power-law, so does the CCDF, with an exponent increased by one. Fitting the tail of the CCDF, or even the entire distribution, results in a major improvement in the quality of fit. An exhaustive discussion of all these issues may be found in [15].

We were able to obtain high quality best fits using three different distribution functions, all compatible with a power-law behavior in the tail. This approach has already been proposed in the literature to explain the power-law in the tail of various software properties [9, 13]. In our study, we are actually interested in answering our research questions, and the power-law behavior in the tail for the distribution of the studied metrics is only a side information, related to the best fitting distribution functions which may eventually best approximate the empirical data. With the best fitting proposed analytical distribution functions, we then try to predict some values for the metrics in the future releases.

The CCDF is defined as 1 − CDF, where the CDF (Cumulative Distribution Function) is the integral of the PDF. Denoting by $p(x)$ the probability distribution function, by $P(x)$ the CDF, and by $G(x)$ the CCDF, we have

$$\begin{aligned} G(x) &= 1 - P(x), \\ P(x) &= p(X \leq x) = \int_{-\infty}^{x} p(x')dx' \\ G(x) &= p(X \geq x) = \int_{x}^{\infty} p(x')dx'. \end{aligned} \tag{1}$$

The distributions we study are a straight power-law—also called Pareto distribution—a log-normal, and a Yule-Simon distribution [21, 34]. The power-law is mathematically formulated as $p(x) \simeq x^{-\alpha}$, where α is the power-law exponent, the only parameter which characterizes the distribution, besides a normalization factor. Since for $\alpha \geq 1$ the function diverges in the origin, it cannot represent real data for its entire range of values. A lower cut-off, generally indicated x_0, has to be introduced, and the power-law holds above x_0. Thus, when fitting real data, this cut-off acts as a second parameter to be adjusted for best fitting purposes. Consequently, the data distribution is said to have a power-law in the tail, namely, above x_0.

The log-normal distribution has been also proposed in the literature to explain different software properties [13, 22, 30]. Mathematically it is expressed by

$$p(x) = \frac{1}{x\sqrt{2\pi\sigma^2}} e^{(\ln x - \mu)^2/2\sigma^2}. \tag{2}$$

It exhibits a quasi-power-law behavior for a range of values and provides high quality fits for data with power-law distribution with a final cut-off. Since in real data largest values are always limited and cannot actually tend to infinity, the log-normal is a very good candidate for fitting power-laws distributed data with a finite-size effect. Furthermore, it does not diverge for small values of the variable, and thus may also fit well the bulk of the distribution in the small values range.

The Yule-Simon distribution is expressed through the Euler Gamma function and has two parameters:

$$\begin{aligned} p(x) &= p_0 \frac{B(x + c, \alpha)}{B(h_0 + c, \alpha)}, \\ B(a, b) &= \frac{\Gamma(a)\Gamma(b)}{\Gamma(a + b)}, \end{aligned} \tag{3}$$

where parameters c and α are derived from the Yule model of the growth of Genera and Species in nature [15, 21, 34]. It produces a distribution with a power-law in the tail with exponent α.

We started the analysis by computing the empirical CCDF's of the software network metrics for the various system studied. The empirical distributions of all considered SNA metrics show the same shape for all releases,both in Eclipse and Netbeans.Therefore,we show only the figures for some selected metrics for the last considered releases of the studied systems, namely, Eclipse-3.4 and Netbeans-6.0.

Figure 1 shows graph and SNA metrics for Eclipse 3.4. All CCDF are reported for convenience in log-log plots. Most CCDF show a small cut-off in the extreme tail, which is typically due to the finite size of the sample. Figure 2 shows the same data for Netbeans 6.0. The behavior of Netbeans metrics is very similar to Eclipse's, with smaller cut-off in the extreme tail, perhaps owing to the higher numbers of CUs.

In order to compare the empirical distributions across the releases, we show in the same plot two SNA metrics, Effective Size and Brokerage, for both Eclipse and Netbeans, to highlight their overlap. Figure 3 shows the persistence of the distributions of these metrics across three different releases, starting from the earliest to the most recent. In Eclipse, the curves slightly differ only in the tail, while in Netbeans they are almost coincident.

The empirical distributions of all considered metrics highly preserve the same shape, meaning that, for each specific metric, a single distribution function may account for the empirical data for all the system releases. Moreover, the distributions of the same metric look also very similar in Eclipse and Netbeans releases. Thus, once this distribution is known for one metric in one release, it is possible to infer the properties of the same metric in other releases, provided that the number of CUs is known.

Regarding what specific distribution function can best fit our empirical data, we experimented with the three distributions cited above—power-law, lognormal, and Yule Simon distributions. Figure 4 shows Fan-in, Fan-out, LOC, Size, and Ties, together with best-fit functions, for Eclipse-3.1. For the LOC metric, only the data with the Yule-Simon best-fit curve is shown, while for the other metrics data and best-fits with all the three distribution functions are shown in two different figures.

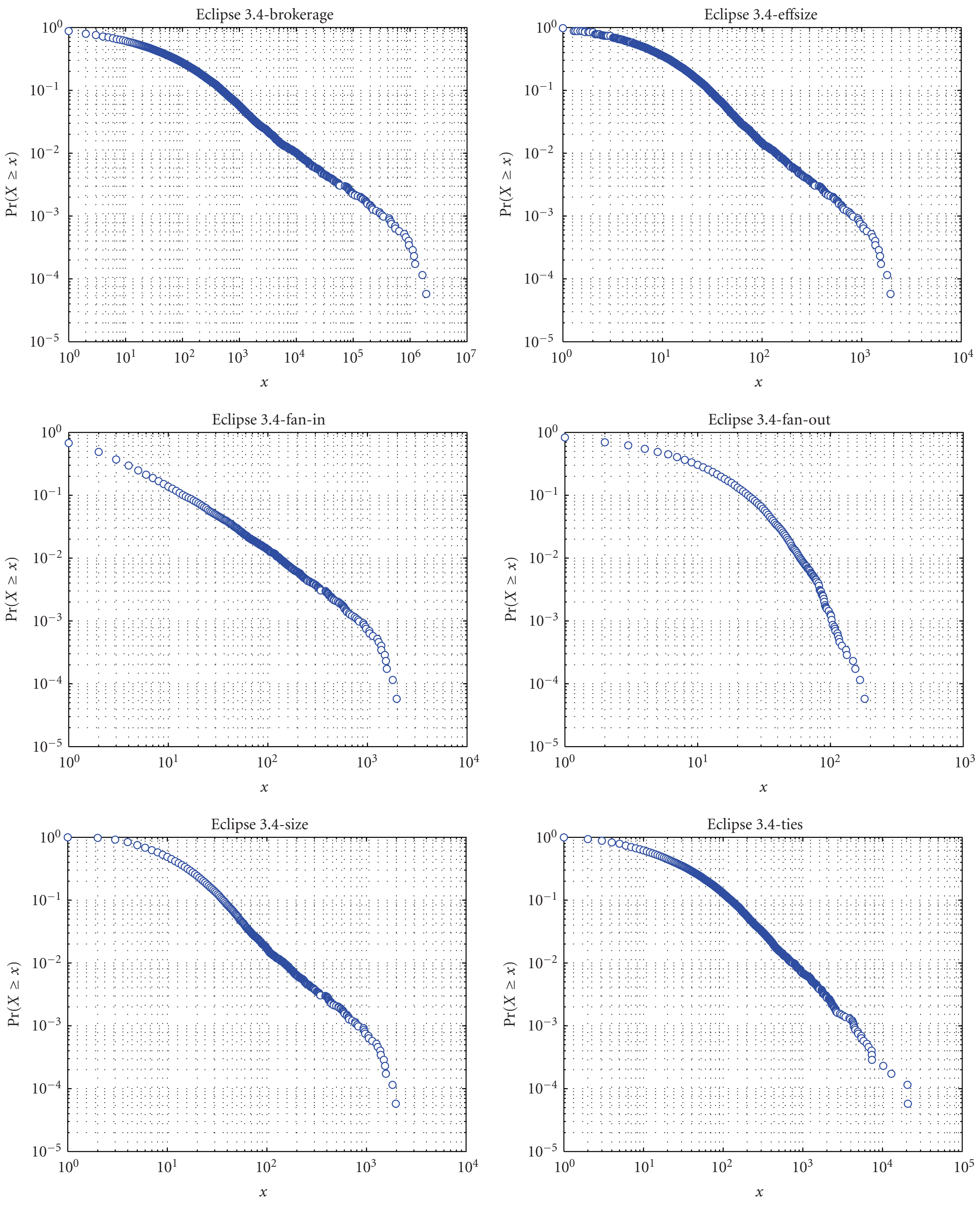


FIGURE 1: CCDF of SNA metrics for Eclipse 3.4 release. The name of the metrics is in the top of the box. The power-law behavior in the tail is patent for all metrics.

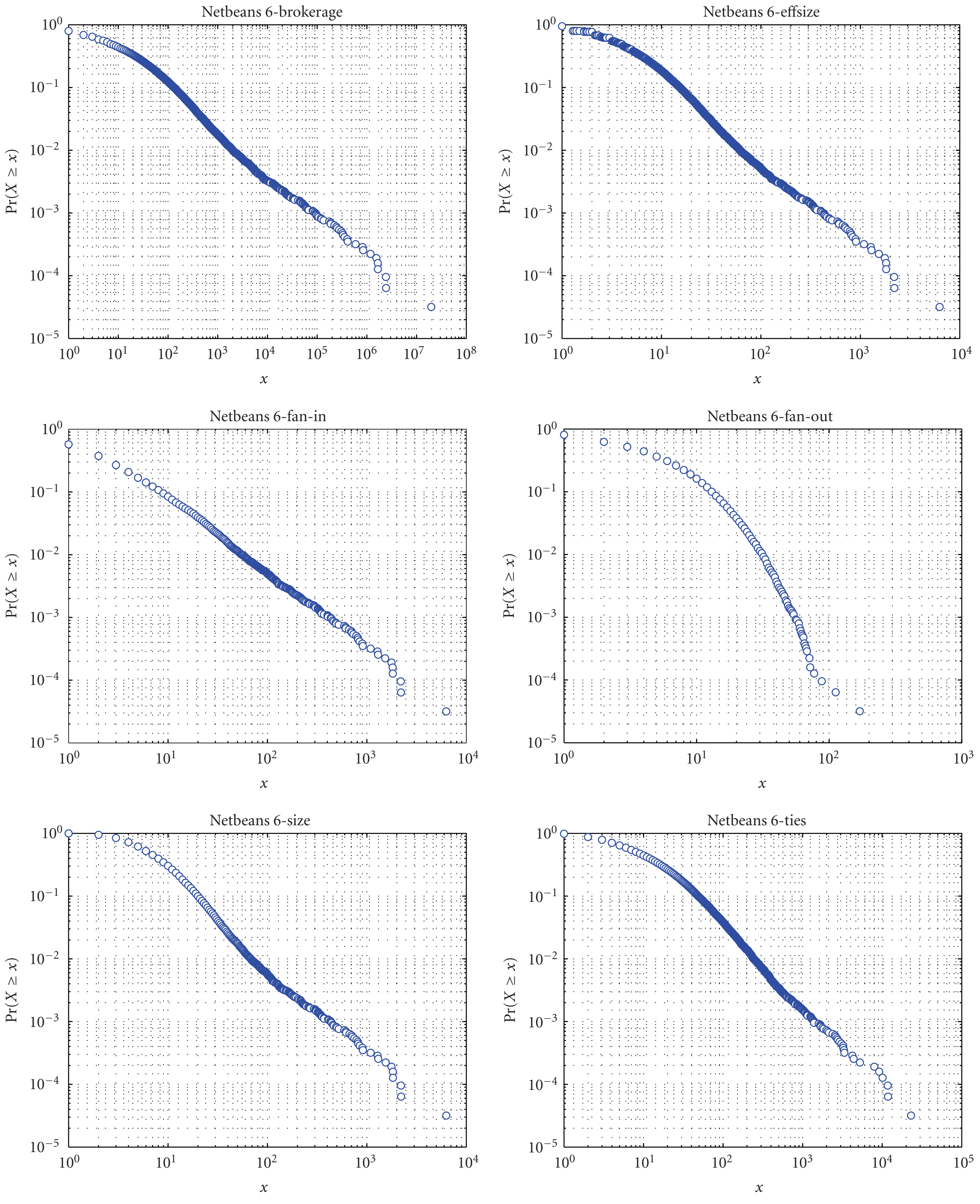


FIGURE 2: CCDF of SNA metrics for Netbeans 6.0 release. The name of the metrics is in the top of the box.

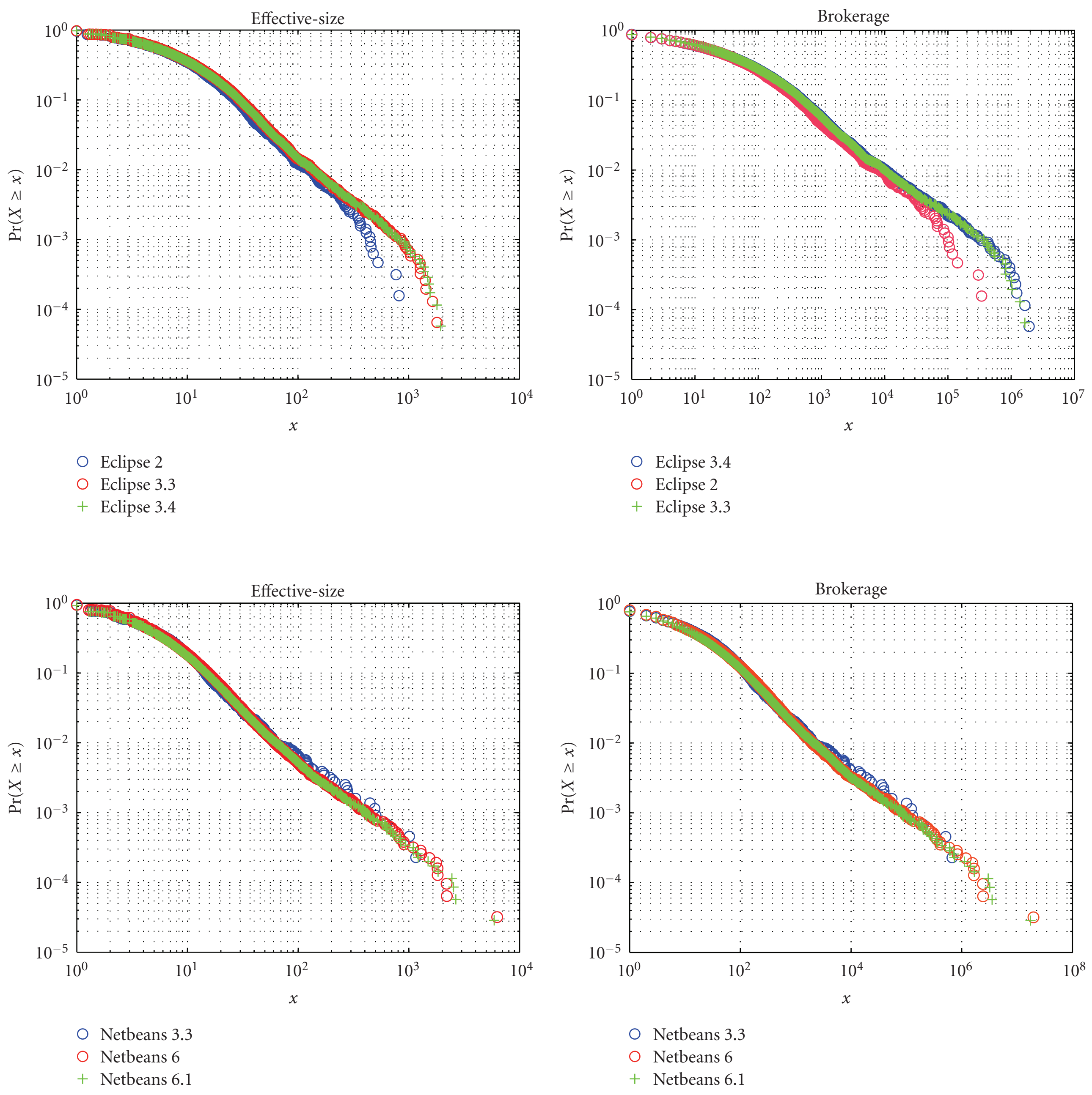


FIGURE 3: CCDF of EffSize and Brokerage metrics for various Eclipse and Netbeans releases. A very similar behavior is patent for all metrics and across all releases of the same system.

The fit using a truncated power-law is almost always very good. Note, however, that this fit is made starting from a minimum value x_0, denoting the value from which the power law tail is apparent. This makes easier to get good fits. The fit with a lognormal is usually the poorest.

This distribution is able to fit very well the bulk of the samples with small values, but in general it tends to zero too quickly with respect to empirical data. The fit with Yule-Simon distribution is sometimes very good, both for small values and in the tails. Other times, it fails to get a good fit in the tail.

In order to evaluate fit accuracy, we used the determination coefficient R^2, defined by $R^2 = 1 - \mathrm{SE}/\mathrm{ST}$, with

$$\mathrm{SE} = \sum_i (f_i - y_i)^2, \qquad \mathrm{ST} = \sum_i (\overline{y} - y_i)^2, \tag{4}$$

where y_i are the empirical CCDF values and f_i the corresponding best fitting values. All the fits have very high determination coefficients, sometimes up to 0.999 (Table 3).

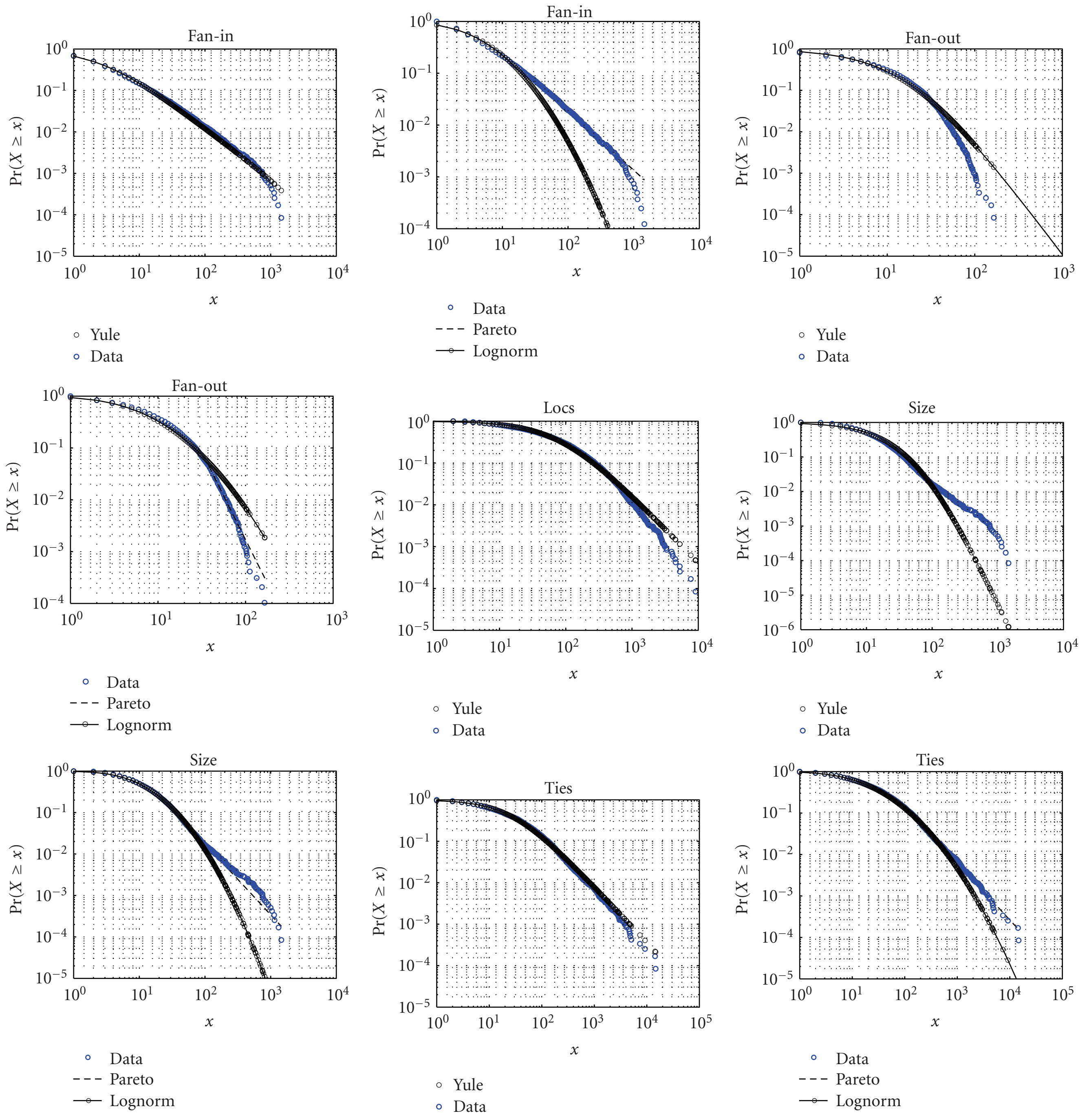


FIGURE 4: Empirical CCDFs of various metrics in Eclipse 3.1, with their best-fit theoretical distributions. Yule-Simon fit is shown separately.

TABLE 3: Determination coefficients for the three distribution functions (Eclipse-3.1).

R^2	Yule-Simon	Lognormal	Power-law
Fan-in	0.999	0.971	0.998
Fan-out	0.995	0.989	0.997
Size	0.987	0.999	0.998
Ties	0.998	0.999	0.999

TABLE 4: Determination coefficients for the three distribution functions (Netbeans-3.2).

R^2	Yule-Simon	Lognormal	Power-law
Fan-in	0.999	0.978	0.998
Fan-out	0.998	0.982	0.996
Size	0.980	0.995	0.998
Ties	0.999	0.998	0.999

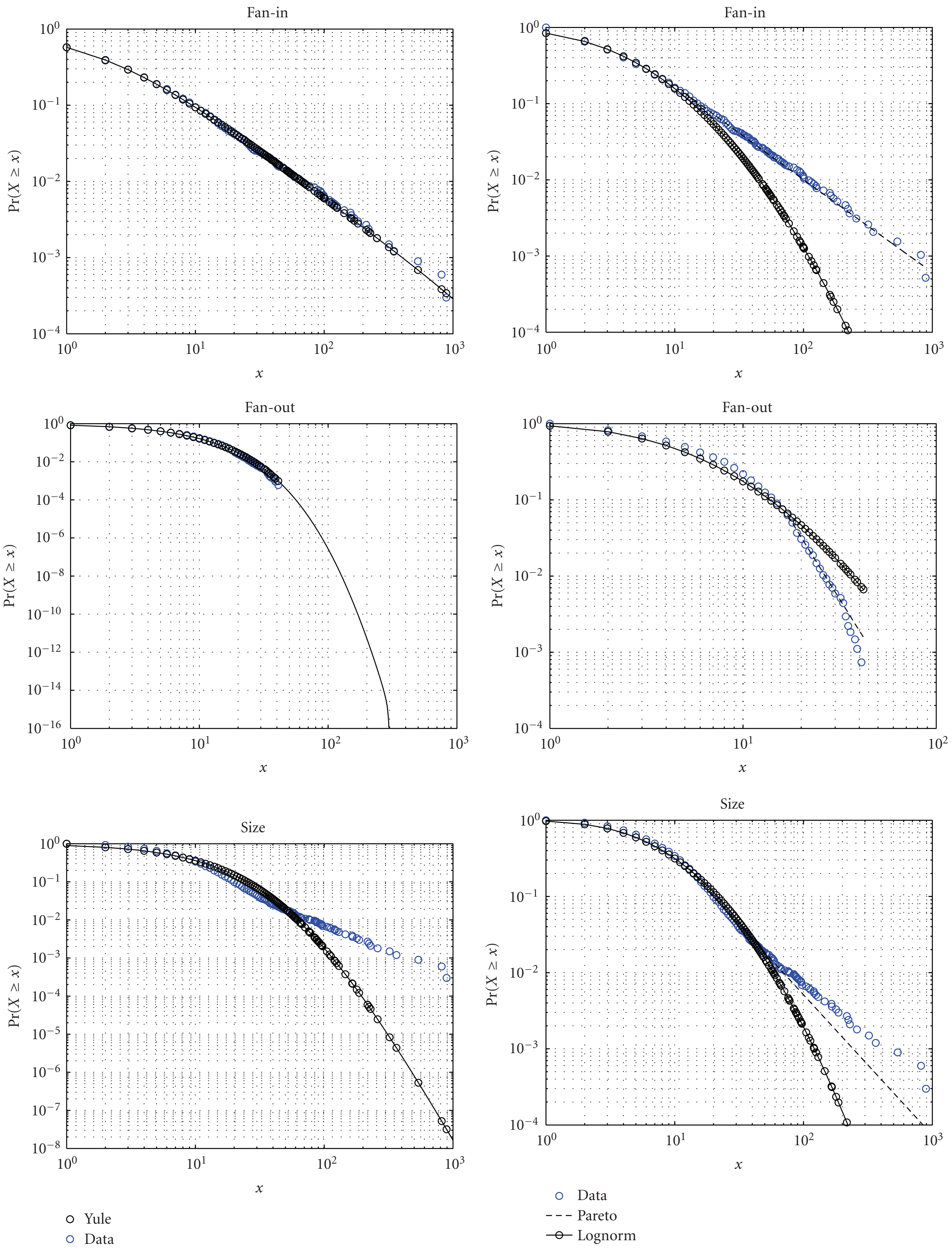
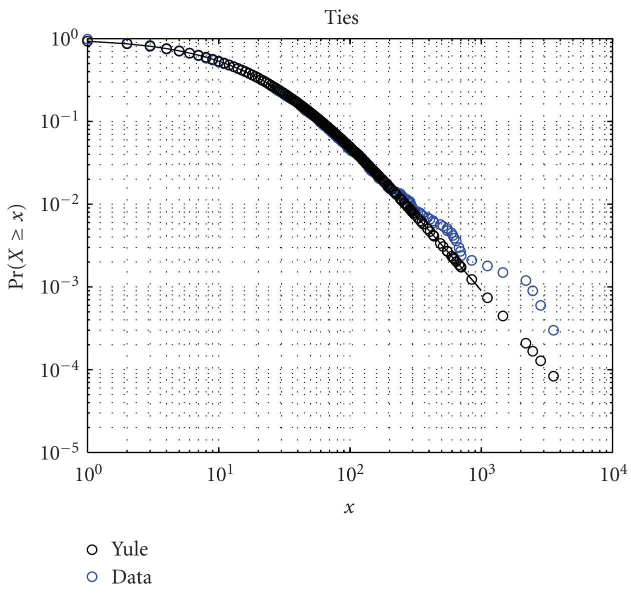


Figure 5: Continued.



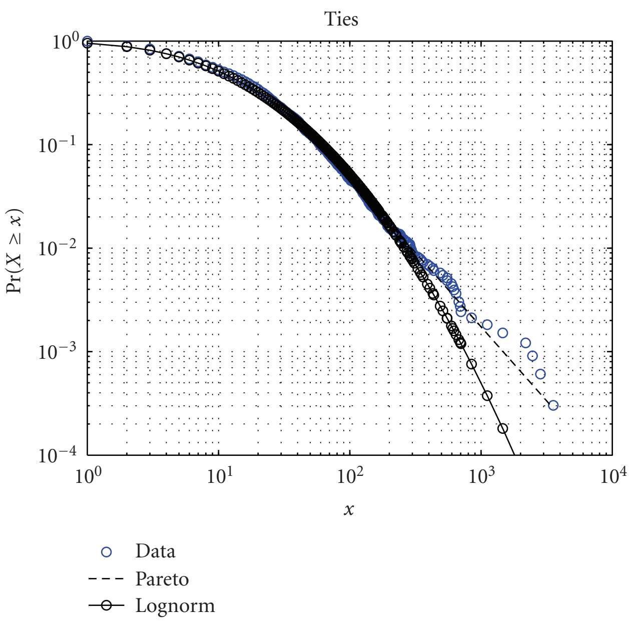


Figure 5: Empirical CCDFs of various metrics in Netbeans 3.2, with their best-fit theoretical distributions. Yule-Simon fit is shown separately.

This suffices to answer to our research questions. It is in fact known that when experimental data are roughly power-law distributed, it is in general extremely difficult to assess the difference among a true power-law and other fat-tail distributions, since typically any statistical test does not rule out one or the other distribution function. In fact, they are often compatible with many different distribution functions [35].

Our purpose in this paper is, on the contrary, to provide a reasonable statistical description of the empirical data, and to find the analytical distribution function with the best fit. This allows us to make statistically reliable forecasts on the value assumed by some metrics in the future system releases. In our case, power-law is not in principle more interesting than the log-normal or Yule-Simon distributions, as long as these provide reliable estimates and good descriptions of the empirical data. Any other statistical speculation in order to discriminate among power-law or other distributions is out of our purposes.

Note that the determination coefficients are evaluated on the linear scale, whereas all the figures are in a log-log scale. In this scale, the discrepancy between best fitting curves and empirical curves is visually enhanced, especially in the tail, whereas in the original scale the fitting curves and the empirical ones visually overlap. On the other hand, our fitting procedure does not rely on any log-log representation of the data.

Figure 5 shows the corresponding data and best fits for Netbeans 3.2. Also for this system the curves provide a very good fitting of empirical data, for the various releases and for the different metrics. Again the coefficient of determination is always close to one (Table 4). The power-law provides an excellent approximation for the data in the tail above the x_0 cut-off, whose value depends on the metrics and on the system version.

The empirical studies presented above answer our first two research questions.

R1: *are there analytical distribution functions describing the empirical data? Have these functions power-law behavior in their tails? What is the significance level of fitting empirical data with these distributions?*

We definitely found that all studied metrics, traditional OO, network-based, and derived from Social Network Analysis, tend to follow precise analytical distributions to a high degree of significance level, according to our best-fitting criteria. These distributions are power-law—from a minimum value of data, x_0—lognormal and Yule-Simon distributions. All three distributions are compatible with a power-law behavior in their tail—regarding the lognormal distribution; this is true for datasets of finite size.

The fit using a truncated power-law are always very good. However, they depend on an *ad hoc* setting of the value x_0, and the power-law regards only the samples whose value $x \geq x_0$. Lognormal distribution shows good fits, according to the value of the determination coefficients, but not as good as power-law. Yule-Simon distribution, on the other hand, shows determination coefficients very similar to those of power-law, but the fit is over all the range of values. So, in general Yule-Simon distribution can be considered the best for most considered metrics.

R2: *are these distributions similar in all the releases and in different systems, or tend to vary significantly?*

We found that all considered metrics have a very consistent statistical behavior across all the releases of the same system, even when these releases span over years and have very different numbers of classes (and CUs).

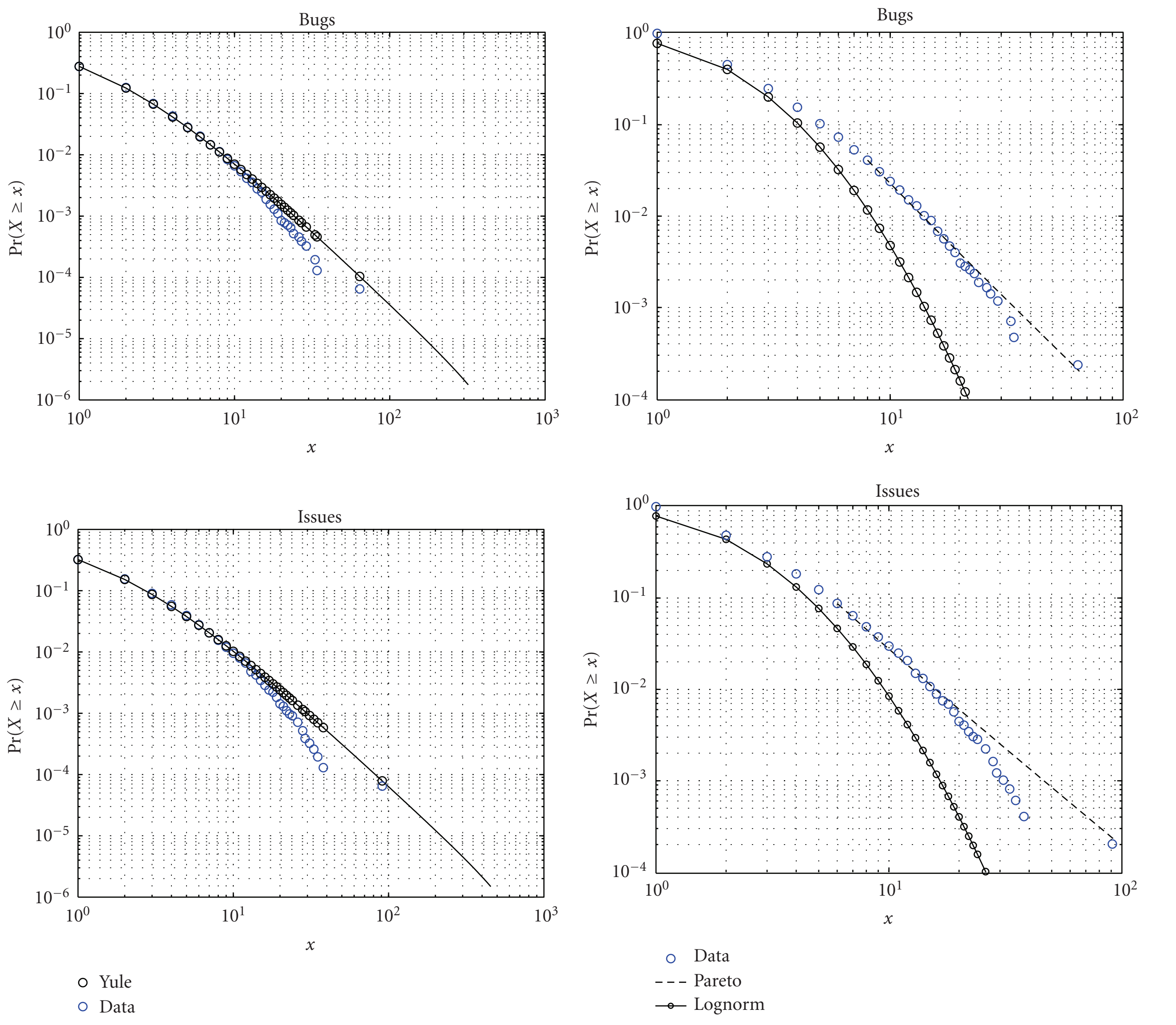


FIGURE 6: Empirical CCDFs of Bugs and Issues in Eclipse 3.3, with their best-fit theoretical distributions. Yule-Simon fit is shown separately.

For completeness, we studied also other Java systems, belonging to the Qualitas Corpus [36] and found that the considered metrics, in systems with over one thousand classes, show behaviors very similar to those reported in this paper for Eclipse and Netbeans.

Next, we analyzed also the metrics related to Issues and Bugs. We found that also the distributions of Bugs and Issues follow similar patterns, in both Eclipse and Netbeans. In Figures 6 and 7, we show the empirical distributions of Issues and Bugs, for the releases 3.3 of Eclipse and 6.0 of Netbeans, together with the best fitting curves of the three considered distribution functions. All Issues and Bugs distributions are very similar throughout all Eclipse and Netbeans releases, so these figures can be considered typical.

The distributions of these metrics are well fitted by the simple power-law, according to the determination coefficient, above a threshold x_0, which depends on the particular data, and very well fitted by the Yule-Simon distribution since the beginning of the data. The log-normal distribution provides a worse fit, even if the determination coefficients R^2 are always above 0.94. Note again that the log-log scale enhances visually the distances in the tail, but the absolute values of the difference among fitting curves and empirical distributions are very small.

8. Correlations

In this section, we report the correlations among SNA metrics, CK metrics, and Bugs. Since the empirical distributions of all metrics are strongly not normal, correlations are better described using the Spearman coefficient. In our study, we computed also Pearson correlations, which are reported only in one case, for comparison. Our considerations, however,

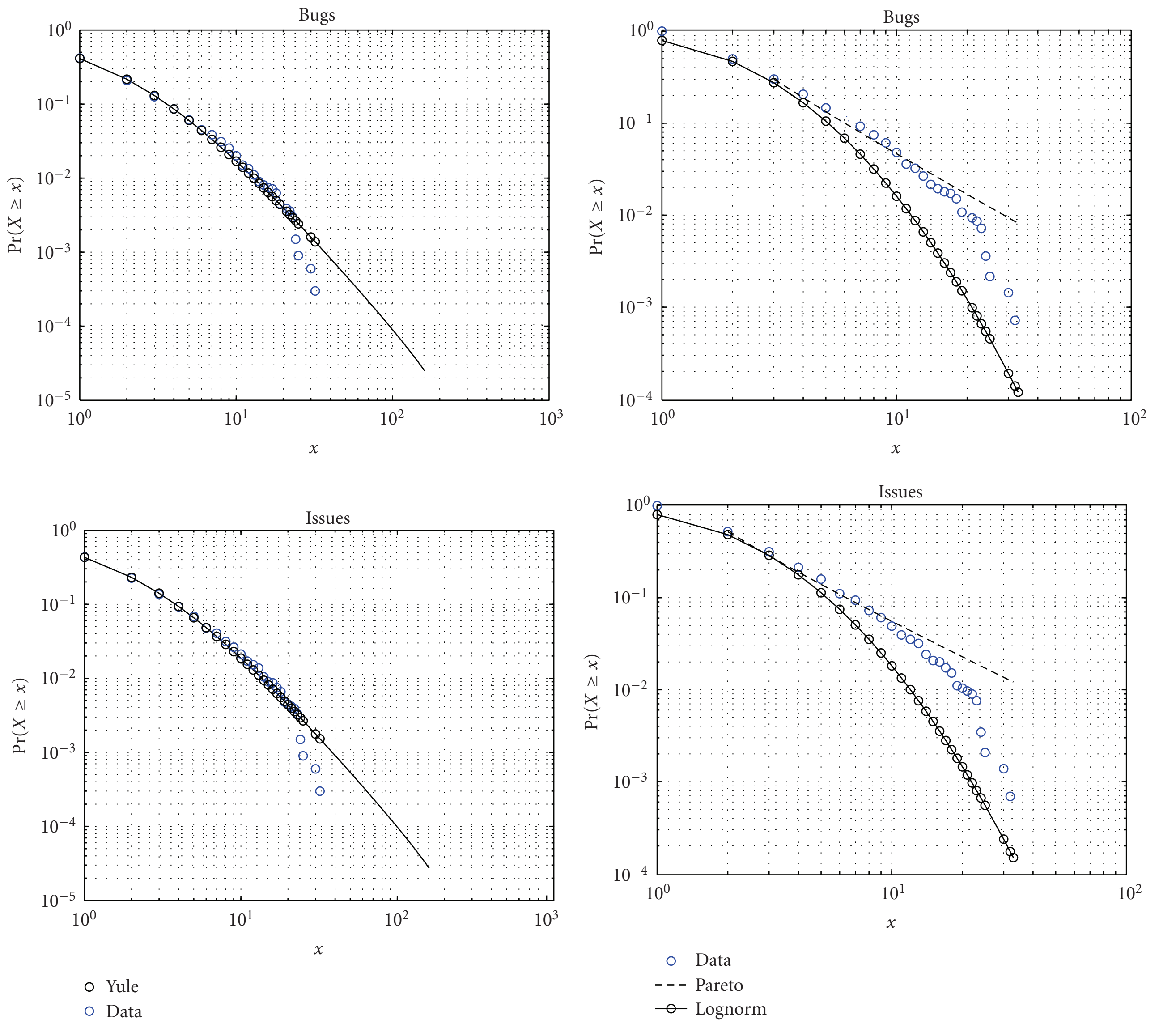


FIGURE 7: Empirical CCDFs of Bugs and Issues in Netbeans 6.0, with their best-fit theoretical distributions. Yule-Simon fit is shown separately.

will refer only to Spearman correlation. Using the latter, data must be ranked, with the correlation coefficient being given by

$$\rho_{SP} = 1 - \frac{6\sum_i d_i^2}{n(n^2-1)}, \tag{5}$$

where d_i are the differences among the ranks of each observation.

We report the correlations only for Eclipse-2.1 and for Netbeans-3.2, as representative of all the other releases. Tables 5 and 6 report correlation data for Eclipse-2.1, using Pearson and Spearman coefficients, respectively. Table 7 reports Spearman coefficients for Netbeans-3.2. The correlation coefficients in all other releases of the same system are substantially similar to those reported here, for both Eclipse and Netbeans.

The higher correlations are among Issues and Bugs, as it is natural, being one a subset of the other. This means that nodes having a high number of Issues also tend to have a high number of Bugs. In other words, the number of Bugs is always about the same fraction of Issues. Thus only one of them will be included in the subsequent analysis.

We computed the correlation matrix among Issue, Bug, CK metrics, LOC, Fan-out, Fan-in, and EGO-metrics. Correlations are almost the same in each release, with fluctuations generally below 10%.

In Eclipse, CK metrics, LOCS, Fan-Out, and EGO metrics generally show a moderate correlation with respect to Issues (Bug). In Netbeans, we have similar correlations, though usually slightly smaller. In both cases, the predictive power of these metrics is similar for the same software system. In both systems, LOC metric is the most correlated

Table 5: Eclipse 2.1. Pearson correlation among metrics.

	Num. issue	Num. bug	LOCS	WMC	RFC	LCOM	CBO	Fan-in	Fan-out	Reach efficiency	Eff.size	Closeness	Dwreach	Infocen-trality	Size	Ties	Nweak-comp
Numbug	97%**	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
LOCS	53%**	53%**		—	—	—	—	—	—	—	—	—	—	—	—	—	—
WMC	49%**	48%**	57%**	—	—	—	—	—	—	—	—	—	—	—	—	—	—
RFC	58%**	59%**	68%**	92%**	—	—	—	—	—	—	—	—	—	—	—	—	—
LCOM	32%**	30%**	19%**	79%**	62%**	—	—	—	—	—	—	—	—	—	—	—	—
CBO	54%**	55%**	65%**	41%**	70%**	11%**	—	—	—	—	—	—	—	—	—	—	—
Fanin	18%**	17%**	62%**	30%**	26%**	26%**	4%**	—	—	—	—	—	—	—	—	—	—
Fanout	51%**	52%**	62%**	30%**	58%**	3%**	94%**	−1%	—	—	—	—	—	—	—	—	—
Reachefficiency	0%	1%	4%**	−4%**	−1%	−3%**	9%**	−14%**	14%**	—	—	—	—	—	—	—	—
Effsize	30%**	29%**	25%**	36%**	40%**	26%**	29%**	96%**	26%**	−10%**	—	—	—	—	—	—	—
Closeness	−2%	−2%	−1%	−1%**	−2%	0%	−3%	−1%	−3%*	−5%**	−2%*	—	—	—	—	—	—
Dwreach	27%**	27%**	23%**	17%**	29%**	4%**	46%**	19%**	50%**	44%**	32%**	−18%**	—	—	—	—	—
Infocentrality	−2%*	−2%*	−2%	−2%	−3%*	0%	−4%**	−1%	−5%**	−61%**	−2%*	94%**	−25%**	—	—	—	—
Size	32%**	31%**	28%**	38%**	42%**	26%**	32%**	95%**	29%**	−10%**	100%**	−2%	34%**	−3%*	—	—	—
Ties	32%**	31%**	27%**	43%**	45%**	37%**	27%**	87%**	23%**	−9%**	89%**	−1%	21%**	−2%	89%**	—	—
Nweakcomp	−23%**	−23%**	−27%**	−18%**	−28%**	−2%*	−39%**	−14%**	−40%**	−2%**	−21%**	4%**	−15%**	5%**	−25%**	−22%**	—
Brokerage	16%**	15%**	9%**	30%**	26%**	35%**	8%**	85%**	3%*	−5%**	83%**	0%	12%**	−1%	82%**	88%**	−7%**

** Correlation is significant at the 0.01 level. *Correlation is significant at the 0.05 level.

Table 6: Eclipse 2.1. Spearman correlation among metrics.

	Num. issue	Num. bug	LOCS	WMC	RFC	LCOM	CBO	Fan-in	Fan-out	Reach efficiency	Effsize	Closeness	Dwreach	Infocen-trality	Size	Ties	Nweak-comp
Numbug	95%**	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
LOCS	46%**	46%**		—	—	—	—	—	—	—	—	—	—	—	—	—	—
WMC	38%**	38%**	84%**	—	—	—	—	—	—	—	—	—	—	—	—	—	—
RFC	46%**	46%**	90%**	89%**	—	—	—	—	—	—	—	—	—	—	—	—	—
LCOM	34%**	34%**	66%**	85%**	74%**	—	—	—	—	—	—	—	—	—	—	—	—
CBO	45%**	45%**	78%**	61%**	86%**	50%**	—	—	—	—	—	—	—	—	—	—	—
Fanin	8%**	7%**	7%**	26%**	7%**	26%**	−14%**	—	—	—	—	—	—	—	—	—	—
Fanout	44%**	44%**	77%**	60%**	84%**	51%**	95%**	−15%**	—	—	—	—	—	—	—	—	—
Reachefficiency	16%**	16%**	29%**	17%**	35%**	13%**	48%**	−29%**	52%**	—	—	—	—	—	—	—	—
Effsize	41%**	41%**	59%**	59%**	68%**	56%**	63%**	45%**	66%**	21%**	—	—	—	—	—	—	—
Closeness	38%**	39%**	51%**	45%**	59%**	42%**	62%**	14%**	66%**	63%**	72%	—	—	—	—	—	—
Dwreach	40%**	40%**	54%**	48%**	62%**	45%**	66%**	15%**	70%**	68%**	76%**	96%**	—	—	—	—	—
Infocentrality	−33%**	−34%**	−45%**	−35%**	−48%**	−35%**	−53%**	−2%*	−55%**	−51%**	−59%*	−79%**	−83%**	—	—	—	—
Size	43%**	42%**	63%**	62%**	71%**	58%**	66%**	46%**	69%**	22%**	98%**	72%**	76%**	−59%**	—	—	—
Ties	42%**	42%**	64%**	60%**	69%**	56%**	65%**	41%**	67%**	17%**	17%**	65%**	69%**	−65%**	94%**	—	—
Nweakcomp	−28%**	−28%**	−48%**	−41%**	−47%**	−37%**	−44%**	−23%**	−44%**	−3%**	−42%**	−31%**	−34%**	51%**	−52%**	−71%**	—
Brokerage	42%**	42%**	61%**	60%**	69%**	57%**	65%**	45%**	67%**	21%**	100%**	73%**	76%**	−59%**	99%**	91%**	—

** Correlation is significant at the 0.01 level. *Correlation is significant at the 0.05 level.

Table 7: Netbeans 3.2. Spearman correlation among metrics.

	Num. issue	Num. bug	LOCS	WMC	RFC	LCOM	CBO	Fan-in	Fan-out	Reach efficiency	Effsize	Closeness	Dwreach	Infocen-trality	Size	Ties	Nweak-comp
Numbug	98%**	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
LOCS	47%**	46%**	%**	—	—	—	—	—	—	—	—	—	—	—	—	—	—
WMC	44%**	42%**	87%**	—	—	—	—	—	—	—	—	—	—	—	—	—	—
RFC	44%**	42%**	87%**	95%**	—	—	—	—	—	—	—	—	—	—	—	—	—
LCOM	41%**	39%**	70%**	87%**	81%**	—	—	—	—	—	—	—	—	—	—	—	—
CBO	33%**	32%**	58%**	53%**	71%**	43%**	—	—	—	—	—	—	—	—	—	—	—
Fanin	13%**	12%**	19%**	34%**	30%**	31%**	13%**	—	—	—	—	—	—	—	—	—	—
Fanout	45%**	44%**	65%**	58%**	68%**	54%**	71%**	3%**	—	—	—	—	—	—	—	—	—
Reachefficiency	17%**	16%**	18%**	14%**	18%**	15%**	18%**	−19%**	49%**	—	—	—	—	—	—	—	—
Effsize	38%**	36%**	52%**	58%**	62%**	54%**	53%**	56%**	70%**	20%**	—	—	—	—	—	—	—
Closeness	38%**	36%**	43%**	43%**	46%**	43%**	38%**	14%**	70%**	73%**	61%**	—	—	—	—	—	—
Dwreach	37%**	35%**	45%**	45%**	48%**	44%**	40%**	17%**	73%**	77%**	66%**	94%**	—	—	—	—	—
Infocentrality	−26%**	−25%**	−30%**	−23%**	−23%**	−24%**	−14%**	9%**	−38%**	−37%**	−26%**	−54%**	−60%**	—	—	—	—
Size	39%**	38%**	55%**	61%**	65%**	57%**	56%**	57%**	73%**	23%**	98%**	64%**	69%**	−28%**	—	—	—
Ties	39%**	37%**	54%**	57%**	60%**	53%**	50%**	48%**	65%**	10%**	83%**	52%**	57%**	−47%**	88%**	—	—
Nweakcomp	−26%**	−25%**	−37%**	−32%**	−32%**	−31%**	−23%**	−16%**	−30%**	12%**	−25%**	−17%**	−20%**	60%**	−36%**	−63%**	—
Brokerage	38%**	37%**	53%**	59%**	63%**	55%**	54%**	56%**	71%**	21%**	10%**	62%**	67%**	−28%**	99%**	84%**	−30%**

** Correlation is significant at the 0.01 level. *Correlation is significant at the 0.05 level.

with Issues. This is expected, because bigger files have a larger chance to produce Issues and Bugs. However, other good predictors of Issues—comparable with LOC—are RFC, Fan-out, Size and, to a lesser extent, LCOM, Ties and Brokerage. In general, we observe that many SNA metrics are quite correlated with the number of Issues (and Bugs), showing the importance of considering these metrics.

In both Eclipse and Netbeans, Fan-in always shows a small—though significant—correlations with Issues. The different correlation between Fan-in and Fan-out with respect to Issues, indicates that to identify a fault-prone node it is important to take into account not only the number of links but also their direction. An Out-link directed from a compilation unit A to a compilation unit B may be considered like a channel easing the propagation of defects from B to A, but not vice versa. This fact highlights the importance of an analysis of a software system as an oriented graph.

CK and LOC metrics correlations with Issues are in line with results previously shown in [5]. In Eclipse, correlations between CK metrics and Eff-Size, Closeness, Size, Ties, brokerage are quite large. Correlations with Nweak-comp, Infocentrality, Dwreach, and Closeness are smaller. Only a minor correlation exists between CK metrics and Reach-Efficiency.

In Netbeans, correlations between CK metrics and Eff-Size, Size, Ties, Brokerage are also large. Smaller correlations hold between CK metrics and Closeness, Nweakcomp, Dwreach. Only minor correlations, like in Eclipse, exist between CK metrics, Reach-Efficiency, and Info-Centrality.

In both Eclipse and Netbeans, the only metrics that are anticorrelated with the number of Issues are Info-Centrality and Nweak-Comp, suggesting that it is better for a CU to have a high Information Centrality and Normalized number of Weak Components, to be less prone to get Issues and Bugs.

Most Eclipse and Netbeans EGO metrics are not strongly correlated with each other. For example, Reach-Efficiency has small correlation with Eff-Size, Size, and Brokerage, and no correlation with Nweakcomp. Size metric is the most correlated with the others EGO-metrics and shows an almost perfect correlation with Eff-Size and Brokerage. Consequently, it is clearly needed to consider just one of these metrics. We suggest to use Size, which is easier to compute and, at least in the considered systems, looks slightly better correlated to Issues.

These findings related to correlations answer our last two research questions.

R4: *are there SNA metrics significantly correlated with software Bugs, and to which extent?*

The data reported, and data very similar to them related to all other considered releases of Eclipse and Netbeans, confirm that there are significant correlations between several SNA metrics and the number of Bugs. These correlations are of the same order of magnitude of more traditional CK metrics—whose predictive power in predicting faulty classes has been studied and assessed for a long time [5, 7]. Note that all CK metrics, and most SNA metrics, are basically complexity metrics, denoting high coupling and/or low cohesion of the measured module. This is consistent with the positive correlation between these metrics and the fault-proneness of the module. However, some SNA metrics are anticorrelated to a fairly high extent with the number of Bugs, and this property might be further studied and exploited.

R5: *are there SNA metrics significantly correlated to traditional CK metrics, and to which extent?*

The study of Tables 6 and 7 confirms that all SNA metrics are significantly correlated to all the four considered CK metrics—WMC, RFC, CBO, and LCOM. Some SNA metrics— namely, Eff-Size, Size, Ties, and Brokerage—show quite high Spearman correlation coefficients with all these CK metrics.

9. Providing Estimates

In this section, we discuss how it is possible to estimate some values for the metrics starting from the knowledge of the analytical fitting functions. We assume that all the data are known for one system release and assume the persistence of the distributions across releases.

Let us consider, for instance, the metric Ties, and the Eclipse releases from 2.1 to 3.3. Let us start with the lognormal distribution. If we compute the estimate of the mean values using the best fitting parameters found, using the usual formula ($\exp(\mu + \sigma^2/2)$), they match actual values with an error of about 15% (see Table 9). With regard to the standard deviation, however, the estimate of the lognormal fails. In fact, empirical data show a systematic increase of their standard deviation, while the lognormal provides a constant value, since the best fitting parameters are almost constant.

It is also possible to estimate the expected maximum value for a lognormal population of finite size n, which depends on n, using the formula [37]:

$$\log(x_{\max}) = \mu + \sigma\sqrt{2\log(n)} - \sigma\frac{(\log\log(n) + \log(4\pi))}{2\sqrt{2\log(n)}} + \epsilon, \tag{6}$$

where ϵ is a small error term. We approximated our estimates using the first two terms, since the third is negligible in our case. The predicted extreme values for the Ties distribution are reported in Table 9, which shows a discrepancy with the empirical values of about 15/20%, which increases with the system size.

If we consider the best-fit power-law distribution, its exponent α_{PL} has always values between 2 and 3, and this is consistent with the power-law property that, for such values of α, the mean is finite, while the standard deviation diverges. In the case of a finite number of samples, this means that the standard deviation has obviously a finite value, but it tends to increase with the number of samples [15]. This is exactly the behavior which we observed. Therefore, when the number of CU increases from a release to another, so does the standard deviation. Note that the power-law cannot fit the bulk of the data, since the cut-off starts at about 140. So, it cannot be used to estimate the mean of the samples.

Table 8: The best fitting parameters for the three different distributions for the metric Ties. For each version of Eclipse, empirical first and second moment, number of CU and maximum value are also reported.

Ties		lognormal		Power-law		Yule-Simon				
Release	μ	σ_{LN}	α_{PL}	x_0	α_{YS}	c	#CU	$\langle x \rangle$	$\langle x^2 \rangle$	
2.1	2.85	1.54	2.38	174	2.23	20.6	7545	59.6	227.3	9799
3.0	2.80	1.54	2.39	141	2.21	19.1	10288	59.2	257.6	11901
3.1	2.85	1.56	2.37	143	2.16	18.6	11854	64.9	294.7	14711
3.2	2.82	1.57	2.35	141	2.14	17.7	14138	65.3	316.2	17029
3.3	2.83	1.57	2.33	145	2.13	17.4	15439	66.8	336.3	18819

Table 9: Estimates for the extreme values of the metric Ties. In the last column, the two values refer to the estimate obtained using parameters from release 2.1, or using parameters from the immediate previous version, respectively.

Release	Actual value	$\langle x_{\max} \rangle_{\text{lognormal}}$	$\langle x_{\max} \rangle$ (Eq. (8), $\alpha_{\text{power-law}}$)	$\langle x_{\max} \rangle$ (Eq. (8), α_{YS})
3.0	11901	12962	12268	12609/==
3.1	14711	13634	13594	14148/14234
3.2	17029	14511	15446	16327/16838
3.3	18819	14967	16463	17539/18363

Using the power-law, however, we may provide an estimate for the maximum value, a quantity more relevant than the estimate of the mean. It is well known that the following formula holds [15]:

$$\langle x_{\max} \rangle \sim n^{1/(\alpha-1)}. \tag{7}$$

So, for two generic releases we can write

$$\frac{\langle x_{\max_1} \rangle}{\langle x_{\max_2} \rangle} = \left(\frac{n_1}{n_2}\right)^{1/(\alpha-1)}, \tag{8}$$

and we can use one extreme value measured from release i to estimate the extreme value of release $i+1$, when CU numbers are known. Using the values in Table 8, the error is about 15%, as reported in Table 9.

The Yule-Simon distribution is a good compromise between the two other considered distributions, because it fits both the bulk and the tail of the data. We numerically estimated the average using the best fitting parameters of the Yule-Simon distribution in Table 8, and they are in agreement with the empirical values. The power-law exponent obtained from the Yule-Simon best fit is among two and three, and it is consistent with the empirical standard deviation, which seems to diverge with the number of CUs. Furthermore, since (7) holds asymptotically, we can use the power-law exponent as obtained from the Yule-Simon best fitting in (8), to estimate the extreme values as before. These are in excellent agreement with the empirical results (Table 9).

We may now answer to the third research question **R3**: *is it possible to use these distributions to estimate the metrics values in subsequent releases?*

We found that mean values, as obtained from the analytical distributions, are in agreement with the empirical ones. From the knowledge of the best fitting parameters of the Yule-Simon distribution in one release, assuming persistence, we estimated the extreme values of subsequent releases using the CU number. Such estimates are in agreement with the empirical values with an error of $\Delta x/x = 456/18819 \simeq 2.5\%$.

These results have been obtained for the metric Ties for Eclipse but similar considerations hold also for the other metrics which are best fitted using Yule-Simon distribution.

10. Conclusions

In this paper, we studied for the first time the distribution of SNA metrics in OO software networks, comparing their properties with those of CK metrics and other graph-related metrics. We used as a central concept the Compilation Unit and not the class, to be able to better study the impact of metrics on Bugs and Issues, which always refer to CUs and not to classes, in commonly used configuration management systems.

The empirical distributions of all the studied metrics systematically present power-laws in their tails. This property holds also for bug distribution. It must be noted that bug distributions may be biased due to the lack of information in CVS commits, thus our results on bug distributions are as reliable as the information about bugs extracted from CVSs. All metrics have very similar features and shapes across all the system releases and also show very similar behavior in both Eclipse and Netbeans systems.

We found analytical distribution functions suitable for fitting the empirical data. Power-law always outperforms other fittings in the tails, whereas Yule-Simon distribution follows the shapes of most metrics empirical distributions very well. In particular, Ties and Fan-in metrics are fitted by Yule-Simon distribution from the very beginning of values, with the determination coefficients being over 0.98. We have shown—using the metric Ties—how it is possible to provide reliable estimates for averages and extreme values of subsequent releases from the knowledge of the best fitting parameters and system size. The knowledge of extreme values of metrics could be exploited to keep under control the quality of software systems, because in general high values of these metrics denote high coupling among classes.

Regarding correlations among SNA metrics and Bugs, they are generally good, and when using the Spearman coefficient to assess them, they are comparable to those of CK metrics. It is known that LOC is one of the metrics best correlated with the number of defects. Nevertheless, as it holds for some other complexity metrics, they focus only on single software elements, while the use of SNA metrics allows to take into account the role of interactions between elements, and how these interactions correlate with defects. Consequently, we can state that the new SNA metrics are worth studying in greater detail, to better assess their predictive power regarding Issues and Bugs, maybe in conjunction, and not as an alternative to more traditional OO metrics.

Future developments of this seminal work will include controlled experiments to better understand the effect of SNA metrics on bug proneness and if they are able to identify different kind of bugs, and the construction of software graphs where the link direction and type are taken into account.

References

[1] S. R. Chidamber and C. F. Kemerer, "Metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[2] F. Brito e Abreu, "The MOOD metrics set," in *Proceedings of ECOOP Workshop on Metrics*, 1995.

[3] M. Lorenz and I. Kidd, *Object Oriented Software Metrics: A Pratical Guide*, Pretience Hall, Englewood Cliffs, NJ, USA, 1994.

[4] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial use of metrics for object-oriented software: an exploratory analysis," *IEEE Transactions on Software Engineering*, vol. 24, no. 8, pp. 629–639, 1998.

[5] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[6] R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, 2003.

[7] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897–910, 2005.

[8] S. Valverde, R. Ferrer Cancho, and R. V. Solé, "Scale-free networks from optimal design," *Europhysics Letters*, vol. 60, no. 4, pp. 512–517, 2002.

[9] G. Concas, M. Marchesi, S. Pinna, and N. Serra, "Power-laws in a large object-oriented software system," *IEEE Transactions on Software Engineering*, vol. 33, no. 10, pp. 687–708, 2007.

[10] A. Potanin, J. Noble, M. Frean, and R. Biddle, "Scale-free geometry in object-oriented programming," *Communications of the ACM*, vol. 48, no. 5, pp. 99–103, 2005.

[11] J. P. Scott, "Social network analysis," *Sociology*, vol. 22, no. 1, pp. 109–127, 1988.

[12] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*, pp. 531–540, May 2008.

[13] P. Louridas, D. Spinellis, and V. Vlachos, "Power laws in software," *ACM Transactions on Software Engineering and Methodology*, vol. 18, no. 1, article no. 2, 2008.

[14] A. Tosun, B. Turhan, and A. Bener, "Validation of network measures as indicators of defective modules in software systems," in *Proceedings of the 1st International Conference on Predictor Models (PROMISE '09)*, 2009.

[15] M. E. J. Newman, "Power laws, Pareto distributions and Zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, 2005.

[16] "Eclipse," http://www.eclipse.org/.

[17] "NetBeans," http://www.netbeans.org/.

[18] "Bugzilla," http://www.bugzilla.org/.

[19] http://netbeans.org/bugzilla/report.cgi.

[20] Cvs, http://www.nongnu.org/cvs/.

[21] H. Simon, "On a class of skew distribution functions," *Biometrika*, vol. 42, pp. 425–440, 1955.

[22] G. Concas, M. Marchesi, S. Pinna, and N. Serra, "On the suitability of Yule process to stochastically model some properties of object-oriented systems," *Physica A*, vol. 370, no. 2, pp. 817–831, 2006.

[23] T. J. McCabe, "Complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, 1976.

[24] S. R. Chidamber and C. F. Kemerer, "Towards a metrics suite for object oriented design," in *Proceedings of the 6th Annual Conference Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '91)*, vol. 26, pp. 197–211, 1991.

[25] C. Andersson and P. Runeson, "A replicated quantitative analysis of fault distributions in complex software systems," *IEEE Transactions on Software Engineering*, vol. 33, no. 5, pp. 273–286, 2007.

[26] H. Zhang, "On the distribution of software faults," *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 301–302, 2008.

[27] R. Vasa, M. Lumpe, P. Branch, and O. Nierstrasz, "Comparative analysis of evolving software systems using the gini coefficient," in *Proceedings of IEEE International Conference on Software Maintenance (ICSM '09)*, pp. 179–188, September 2009.

[28] S. Valverde and R. V. Solé, "Network motifs in computational graphs: a case study in software architecture," *Physical Review E*, vol. 72, no. 2, Article ID 026107, pp. 1–8, 2005.

[29] R. Wheeldon and S. Counsell, "Power law distributions in class relationships," in *Proceedings of the 3rd IEEE International Workshop onSource Code Analysis and Manipulation (SCAM '03)*, 2003.

[30] G. Baxter, M. Frean, J. Noble et al., "Understanding the shapeof Java software," in *Proceedings of the 21st ACM SIGPLAN Conference on Object-Oriented Programming Languages, Systems, and Applications (OOPSLA '06)*, Portland, Ore, USA, October 2006.

[31] G. Concas, M. Marchesi, A. Murgia, S. Pinna, and R. Tonelli, "Assessing traditional and new metrics for object-oriented systems," in *Proceedings of the Workshop on Emerging Trends in Software Metrics (ICSE '10)*, Cape Town, South Africa, May 2010.

[32] M. Eaddy, T. Zimmermann, K. D. Sherwood et al., “Do crosscutting concerns cause defects?” *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 497–515, 2008.

[33] K. Ayari, P. Meshkinfam, G. Antoniol, and M. Di Penta, “Threats on building models from cvs and bugzilla repositories:the mozilla case study,” in *Proceedings of the 17th Annual International Conference on Computer Science and Software Engineering (CASCON ’07)*, Toronto, Canada, October 2007.

[34] G. Yule, “A mathematical theory of evolution based on the conclusions of dr. j.c. willis,” *Philosophical Transactions of the Royal Society London B*, vol. 213, pp. 21–87, 1925.

[35] A. Clauset, C. R. Shalizi, and M. E. J. Newman, “Power-law distributions in empirical data,” *SIAM Review*, vol. 51, no. 4, pp. 661–703, 2009.

[36] Qualitas Research Group, “Qualitas Corpus Version 20090202,” The Universityof Auckland, February 2009, http://www.cs.auckland.ac.nz/ewan/corpus.

[37] N. D. Singpurwalla, “Extreme values from a lognormal law with applications to air pollution problems,” *Technometrics*, vol. 14, no. 3, pp. 703–711, 1972.