Connectivity properties of the Apache Ant class collaboration network

Miloš Savić, Mirjana Ivanović, and Miloš Radovanović

Abstract—Properties of networks whose nodes represent classes of an object-oriented (OO) system, and links denote dependencies between them, are fundamental to understand, describe and characterize software complexity at the structural level. In this paper, degree distributions of the class collaboration network that represents class dependencies of Apache Ant software were examined. In contrast to similar studies where complementary cumulative degree distributions were tested only against a power law, here we also consider the exponential distribution. Our analysis revealed scale-free phenomena in in-degree and total degree distributions. However, Ant’s out-degree sequence is better modeled with the exponential distribution than a power law. Based on this result, implications relevant to software engineering are discussed in the aspect of the absence of a characteristic scale of class reuse and the presence of a characteristic scale of class aggregation.

I. INTRODUCTION

In the past decade, a huge body of research has investigated statistical properties of networks that represent various complex biological, social, technological, and conceptual systems. Those analyses had revealed some common properties of real-world networks such as the existence of nodes with extremely high degree compared to the average (hubs), heterogeneous connectivity structure characterized by a power law (the scale-free property), small distance between two randomly chosen nodes (the small-world property), and highly dense sub-graphs induced by a randomly chosen node and adjacent nodes (high level of clustering) [1–3]. Recent research [4–10] suggests that networks formed from large and widely used software systems share characteristics common to many analyzed real-world networks.

A large program written in an object-oriented (OO) programming language is typically divided into a set of classes. Classes defined in an OO software system can be viewed as nodes, and dependencies between them as links in a class collaboration network (CCN). The links in a CCN are directional, two classes A and B are connected by the directed link A → B if class A references class B. Class collaboration networks can be viewed as simplified class diagrams, a notion extensively used in object-oriented design, because they preserve only relations between classes and discard other types of information about nodes (classes) and links (relations) presented in class diagrams.

Modern OO software systems are collections of many hundreds or even thousands of classes. Traditionally, the complexity OO software systems is viewed as the internal complexity of classes defined in some OO program and estimated using software metrics such as the lines of code (LOC) metric or cyclomatic complexity. However, there is another level of complexity in OO systems – structural complexity of dependencies between classes. To illustrate the differences between these two layers of complexity, we will give two simple examples. Let us suppose that we need to implement an OO system which has to realize some complex requirements specification. Such an OO system can be implemented by only one class, and the internal complexity of that class will be enormously high, but the complexity of the appropriate dependency structure will be the simplest possible network (a network that contains just one node and no links), which means that the structural complexity of the system is minimized. On the other side, good decomposition of functionality can lead to a set of classes whose internal complexity is minimized, but the appropriate network which reflects the structure of the system will be enormously complex (comparing to the previous case, when the structural complexity is minimized), and not easy to comprehend.

The following simple example shows that reducing complexity at one level increases complexity at the other level. Let us suppose that we have an OO system with three defined classes A, B and C, where class A references classes B and C, and we will suppose that classes B and C have a relatively large portion of identical code. In such a situation, a refactoring of the system can be done: duplicated code can be isolated into a new class D which will be referenced by classes B and C. After the refactoring internal complexity of the classes is reduced (both average and total LOC are smaller), but the structure of dependencies becomes more complex, because the refactoring process introduced one more node, two more links and the change in class collaboration network topology (before the process, the class collaboration network was a tree, and after it is a network which contains an undirected loop).

Statistical properties of class collaboration networks can help us to understand and quantify the organizational structure of corresponding software. The degree distribution is a widely used measure of network complexity [1–3]. The degree of a node is the number of links connected to it, and degree distribution $P(k)$ indicates the fraction of nodes that have exactly $k$ links. One of the main advantages of the degree distribution is its discriminative power. Static random networks (Erdős-Rényi random graphs [12] and
Watts-Strogatz small-world networks [13]) have bell-curve shaped degree distributions, evolving random networks with the uniform attachment principle are characterized by the exponential distribution \( P(k) \sim \exp(-\beta k) \), while degree distributions of evolving random networks with the preferential attachment principle follow the power law \( P(k) \sim k^{-\gamma} \) [14]. For static random networks, the set of nodes is definitely formed before any connection is established. In contrast, evolving random networks grow over time by adding new nodes in each evolutionary step. The attachment principle determines how newly created nodes are integrated into the network. In the case of the uniform attachment principle, each node has equal probability to establish a connection with a newly added node, while the preferential attachment principle is based on the idea that the connection probability depends on node degree (nodes with higher degree have a higher probability to be connected with the newly created node).

In a directed network, the degree of a node is the sum of its in-degree (number of links entering the node) and out-degree (number of links exiting the node), thus there are three degree distributions that describe the connectivity of a network: total degree distribution (or just degree distribution) \( P(k) \), in-degree distribution \( P_{in}(k) \), and out-degree distribution \( P_{out}(k) \), where \( P_{in}(k) \) and \( P_{out}(k) \) indicate fractions of nodes that have exactly \( k \) in-coming and out-going links, respectively.

In this work, degree distributions of the CCN associated with the Apache Ant software system are examined. The network was extracted from the source code of Ant version 1.7.0 using the Yaccne tool (detailed description of this software can be found in [10]). In Section II, motivation for this study and its contributions are highlighted. A representative selection of related works, which view software systems as complex networks is reviewed in Section III. The following section describes the results obtained in the analysis of the degree distribution of Ant’s CCN. In Section V, the results are discussed from the software engineering perspective. Finally, in Section VI we give the conclusion.

II. MOTIVATION AND CONTRIBUTIONS

In our previous study [10], we investigated complementary cumulative degree distributions of five class collaboration networks formed from software systems written in Java, among them Ant. Compared to [10], in this study two improvements are made. Firstly, degree distributions are calculated for the largest weakly connected component. Small isolated clusters of trivial complexity which increase \( P(k) \) for small values of \( k \) are omitted from analysis. In this case, observed degree distributions describe the connectivity of the largest fraction of connected nodes, which is the sub-structure that carries complexity. Secondly, the exponential distribution, characteristic of evolving random networks that employ the uniform attachment principle, is considered as a candidate to model in, out and total degree sequences. In related work [4–11], which examined degree distributions of various types of collaboration networks formed from software systems, degree distributions were tested only against a power law. Software systems evolve over time and it is natural to suppose that software networks have a topology which can be modeled by evolving random networks. However, a power law cannot characterize all types of evolving random networks, only those based on the preferential attachment (or similar) principle. In this work, as the main result which provides completely new insights related to software engineering, we will show that Ant’s complementary cumulative out-degree distribution cannot be considered scale-free. In more detail, this means the following:

1) Ant’s complementary cumulative out-degree distribution has a power-law scaling region which does not cover the whole range of out-degree values (as was already reported in [10]).
2) The complementary cumulative distribution function of the exponential distribution better models the observed out-degree sequence through the whole range of out-degree values than a power law over a power-law scaling region.

In [10], CCNs extracted from ten successive versions of Ant (from version 1.5.2 to version 1.7.0) were compared, and it was shown that the preferential attachment principle can explain changes through Ant software evolution at the class collaboration level for in-coming links. This result is significant because it suggests that the preferential attachment principle is relevant to software evolution (and this relevance was also shown in [11] using the Linux kernel as a case study). This result also explains how Ant’s complementary cumulative in-degree distribution evolved into a scale-free state. Although in [10] it was shown that Ant’s complementary cumulative out-degree distribution reveals a region that follows a power law, we were unable to obtain evidence of preferential attachment through evolutionary changes of out degrees. Exponential scaling of Ant’s complementary cumulative out-degree distribution explains why: this type of scaling does not characterize the preferential attachment, but it does signify the uniform attachment principle.

III. RELATED WORK

Valverde et al. [4] analyzed the JDK class collaboration network extracted from the class diagram. They found that two largest connected components in the JDK network are scale-free (with gamma exponents 2.5 and 2.65, respectively). However, in their analysis link directionality was ignored.

The same authors [6] analyzed the JDK class collaboration network, this time represented as a directed graph, again extracted from the class diagrams. They obtained a power-law in unnormalized complementary cumulative in-degree and out-degree distributions for the largest weakly connected component and the second largest weakly connected component. They also analyzed giant weakly connected components contained in collaboration networks extracted from the source code of 18 C/C++ applications. For all networks they obtained that in/out complementary cumulative degree
distributions follow power-law with $\gamma_{in}$ in range 1.94 to 2.55 and $\gamma_{out}$ in range 2.41 to 3.39.

Myers [5] examined collaboration networks associated with three open source object-oriented (OO) systems written in C++ (VTK, DM, AbiWord) and three written in C (Linux, MySQL, XMMS). He computed the unnormalized complementary cumulative degree distributions for in-coming and out-going links and found that these distributions reveal a power-law scaling region. Values of $\gamma_{in}$ are in range [1.9, 2.5], of $\gamma_{out}$ in range [2.4, 3.33] and $\gamma_{out} \geq \gamma_{in}$ for each analyzed network. Myers also developed a simple model of software evolution based on two refactoring techniques (decomposition of large functions into a set of smaller ones and removal of duplicated code).

De Moura et al. [7] investigated collaboration networks of four open source software projects (Linux, XF86, Mozilla and Gimp) extracted from source code by parsing C/C++ header files. They omitted link directionality and analyzed collaboration networks as undirected graphs. They computed unnormalized degree distributions and found scale-free, small-world and larger than random graph clustering coefficients.

Potanin et al. [8] analyzed run-time object graphs (dynamic, run-time analogues of static collaboration networks) of several OO systems (Java ArgoUML, Java BlueJ, Java Forte, Java Jinsight, Java Satin, C++ GCC, Self and Smalltalk). Their research confirmed power laws in the in-degree and out-degree distributions.

Hyland-Wood et al. [9] analyzed class collaboration networks of two open source software projects written in Java for a fifteen-month period of development and produced collaboration graphs at package, class and method levels. The collaboration graphs were found to form networks which exhibited approximately scale-free properties at all three levels during each analyzed period.

Common to all mentioned works is that degree distributions were tested only against a power-law.

IV. RESULTS

Connectivity properties of a directed complex network can be summarized by in, out and total degree distributions, denoted as $P_{in}(k)$, $P_{out}(k)$ and $P(k)$, respectively. Degree distributions of random growing networks have some nice properties which make them easy to identify. If $P(k)$ (the same holds for $P_{in}(k)$ and $P_{out}(k)$) follows the power law $P(k) \sim k^{-\gamma}$ then the plot of $P(k)$ on logarithmic scales appears as a straight line with slope $-\gamma$ ($\log P(k) \sim -\gamma \log k$). This gives us a simple way to determine if a degree distribution follows the power law. However, it is better to examine the complementary cumulative distribution (CCD) instead of the degree distribution, because the CCD reduces noise that may appear in the tail of empirically observed degree distributions, and gives more accurate estimations of the scaling exponent $\gamma$ [15]. Complementary cumulative degree distribution $CCD(k)$ is the probability of finding a node with degree greater than or equal to $k$, that is, $CCD(k) = \sum_{i=k}^{\infty} P(i)$.

If $P(k) \sim k^{-\gamma}$ then $CCD(k) \sim k^{-(\gamma-1)}$ [2], therefore the plot of $CCD(k)$ on log-log scales also appears as a straight line with slope $-(\gamma-1)$. In the case that $P(k)$ is exponentially distributed, $P(k) \sim \exp(-\beta k)$, then $CCD(k)$ is exponential with the same exponent $\beta$, $CCD(k) \sim \exp(-\beta k)$ [2]. When plotted on semi-logarithmic scales both functions appear as straight lines with slope $-\beta$ (if $f(k) \sim \exp(-\beta k)$ then $\log f(k) \sim -\beta k$). This makes power-law and exponential degree distributions particularly easy to spot by plotting the corresponding complementary cumulative degree distributions on logarithmic scales (for power laws) or semi-logarithmic scales (for exponentials).

The described plotting techniques are used in the analysis of observed complementary cumulative degree distributions associated with Ant CCN.

We computed complementary cumulative in, out and total degree distributions for the largest weakly connected component of Ant CCN (97.81% of nodes) and tested them against the power law of the form $ax^\gamma$ and complementary cumulative distribution function (CCDF) of the exponential distribution (CCDF of the exponential distribution is of the form $e^{-\beta x}$). Coefficient of determination ($R^2$) was used as a goodness of fit measure. Figures 1, 2, 3 show in, out and total CCD for the Ant class collaboration network with fitted power law and CCDF of the exponential distribution. It can be seen that the power law nearly ideally fits in and total CCDs. Estimated values for scaling exponents are $\gamma_{in} = 2.05 \pm 0.01$ and $\gamma_{out} = 2.51 \pm 0.02$ (slopes of the fitted lines are $-1.05$ and $-1.51$, respectively). In contrast, CCDF of the exponential distribution better fits out-CCD than a power law. Although out-CCD in the tail (for out-degree $> 5$) has a power-law scaling region followed by faster decay (for out-degree $> 21$), the fit of exponential CCDF, which covers the whole range of out-degree values, has a greater $R^2$ value ($R^2$(power-law fit) = 0.99181, $R^2$(exponential fit) = 0.99407). Also, it can be visually seen that on semi-log scales out-CCD appears as a straight line with small fluctuations (Figure 4).

V. IMPLICATIONS RELEVANT TO SOFTWARE ENGINEERING

Software engineering practice and simple software metrics such as LOC can give us evidence that modern software systems are complex artifacts. Traditional software metrics used for estimating software complexity are mainly oriented to software entities (functions, classes, methods). These metrics view software as a collection of isolated entities and derive overall software complexity as an average or cumulative complexity of the entities. Also, they can be used to identify complex entities in order to refactor them into smaller, less complex entities. Although this process reduces complexity of entities, it increases complexity at a higher, structural level: new entities and new dependencies between entities appear. Recent progress in the theory of complex networks enables us to measure and characterize software complexity at the structural level, where software
is not seen as a set of isolated, but rather of interconnected software entities.

In-coming degree of a node in a class collaboration network reflects the degree of reuse of a class (or interface) represented by the node. The existence of a power law in-degree distribution implies a broad spectrum of code reuse: most of the classes present in Apache Ant (65.59%) are not reused, but there is a statistically significant number of highly reused classes (2.23%), whose degree of reuse is far above average. For example, two most reused classes “BuildException” and “Project” have 459 and 313 in-coming links, respectively, and average in-degree is 4.768. Preferential attachment, a generating mechanism for power-law degree distributions and empirically verified in Ant for in-coming links [10], suggests the tendency of increasing reuse for already highly reused classes as the class collaboration network grows through software development. Software engineering practice encourages code reuse and this tendency may seem very desirable. However, it is very difficult to change highly reused classes because of their importance to the stability of the software system. Changes in a highly reused class may affect a large number of classes which directly or indirectly depend on it. Thus, identifying highly reused software entities, especially ones which do not realize simple functionalities, may help to prevent the following conflicted situation: the presence of highly reused, hard-to-maintain classes with the tendency of increasing reuse which makes them even more difficult to maintain.

Our findings obtained though the conducted experiments suggest that out-degree complementary cumulative distribution of Apache Ant is better modeled with exponential complementary cumulative distribution functions than a power law. Power law for out-going links was reported in studies which analyzed software networks without ignoring
link directionalities [5, 6, 9, 10], but where complementary cumulative distributions were tested only against a power law. Out-going degree of a node in a CCN reflects the degree of aggregation of the class represented by the node. As noted by Myers [5], classes that have high out-degrees are intra-complex because they aggregate behavior from many other classes. To confirm this thesis we investigated how internal complexity of Ant classes estimated using traditional software metrics is correlated with out-degrees of appropriate nodes in the Ant class collaboration network. Figure 5 shows average WMC (weighted method per class, which is the sum of cyclomatic complexity of methods contained in a class) as a function of out-degree for Ant classes. A trend of increasing intra-complexity (estimated using WMC) can be noticed as out-degree increases. The same trend appears when the intra-complexity of classes is estimated using the LOC (lines of code) metric (see Figure 6). In contrast to the in-degree distribution, the absence of a heavy-tailed out-degree distribution implies the absence of a broad spectrum of intra-complexities. This means that software systems exhibit characteristic intra-complexity (average out-degree) which can describe well the overall intra-complexity of the software system.

Fig. 5. Average WMC as a function of out-degree for Ant classes

Tables I and II show top ten highest in-degree and out-degree nodes present in Ant class collaboration network, respectively. As can be seen class “Project” appears in both tables, which means that this class exhibits a high degree of reuse and aggregation at the same time. It has both significant internal complexity (due to aggregating the behavior of several other classes) and significant external responsibility (because it is used in a lot of other classes defined in the Ant software project). It is interesting to observe that in-degree and out-degree measures are very similar to the information-flow metrics defined by Henry and Kafura [16]: fan-in (the number of functions calling a given function) and fan-out (the number of functions being called from a given function). Actually, in-degree and out-degree are fan-in and fan-out analogues at the class collaboration level. Henry and Kafura [16] defined a complexity metric $C_p$ calculated as $C_p = LOC \ast (\text{fan-in} \ast \text{fan-out})^2$, where LOC represents the number of lines of code in a function. According to Zimmermann et al. [17], functions with a large value of Henry-Kafura complexity may indicate poor design and such entities are candidates for refactoring. If Henry-Kafura complexity for classes is defined as $C_p = LOC \ast (\text{in-degree} \ast \text{out-degree})^2$, then class “Project” has the highest Henry-Kafura complexity and its $C_p$ value is at least two orders of magnitude higher than others ($C_p$ values for the top ten in-degree/out-degree nodes are shown in Tables I and II).

Power laws in in-degree distributions suggest a trend of increasing reuse of highly reused nodes, and therefore good predictions can be made about classes that will be reused by classes added as software evolves. In contrast, the uniform attachment principle, a generating mechanism for the exponential degree distribution observed in out-going links, suggests that prediction about aggregation cannot be made: statistically speaking, each class has the same a priori probability to aggregate newly added classes. In other words, based only on the topology of a CCN, reuse of classes can be predicted, but aggregation cannot.
VI. CONCLUSION

This paper investigated complementary cumulative degree distributions of the largest weakly connected component of Ant’s class collaboration network. Distributions were tested against a power law, characteristic of scale-free networks, and the complementary cumulative distribution function of the exponential distribution, characteristic of evolving random networks with the uniform attachment principle. It was shown that Ant’s network exhibited the scale-free property for in and total degrees, but out-degree sequence can be better modeled with the exponential distribution. On the basis of this, implications relevant to software engineering were discussed in the following aspects: importance of determining nodes with large in-degree that correspond to highly reused classes, characteristic scale of intra-complexities among classes, predictability of class reuse, and unpredictability of class aggregation.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of this work by the Serbian Ministry of Education and Science through project Intelligent Techniques and Their Integration into Wide-Spectrum Decision Support, no. OI174023.

REFERENCES