ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2011, Vol.40, No.1

CHARACTERISTICS OF CLASS COLLABORATION NETWORKS IN LARGE JAVA SOFTWARE PROJECTS

Miloš Savić, Mirjana Ivanović, Miloš Radovanović

Department of Mathematics and Informatics, Faculty of Science, University of Novi Sad
Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia
e-mail: milsav@gmail.com, mira@dmi.uns.ac.rs, radacha@dmi.uns.ac.rs

crossref http://dx.doi.org/10.5755/j01.itc.40.1.192

Abstract. Understanding software structural complexity and evolution plays an important role in controlling the software development and maintenance process. Recent studies have shown that the theory behind complex networks, especially the theory of scale-free networks, can be a useful approach to the analysis of concrete software systems. In this paper, class collaboration networks associated with five large Java software systems (JDK, Ant, Tomcat, Lucene and JavaCC) are analyzed in order to determine whether they belong to the class of scale-free networks, and examine their small-world characteristics. For each analyzed network, we detected (approximately) scale-free and (ultra) small-world properties. The results indicate that general conclusions from scale-free network theory can be applied to Java software systems in order to understand their structural complexity and model software evolution at the structural (class collaboration) level. Moreover, we examine class collaboration network evolution of Ant, in order to check the preferential attachment hypothesis of the Barabási-Albert model. For several major Ant network transitions we conclude that preferential attachment can successfully model Ant evolution at the class collaboration level. Finally, we discuss the implications of our results on software engineering, in several aspects: identification of important classes/interfaces, software testing strategy, and efficient communication among software entities.

Keywords: collaboration network, Java, scale-free, small-world, software evolution.

1. Introduction

In the past decade, researchers have investigated a variety of real-world systems modeled as complex networks, which connect a number of subsystems and components using different types of relations. Empirical studies of existing complex networks (e.g., the World-Wide-Web, collaborations in science, ecological networks, metabolic networks, power networks, etc.) have shown that these networks, despite representing different types of systems, share many common features such as power-law degree distribution, the small-world property, and high levels of clustering. These discoveries inspired a theory of complex networks which focuses on principles by which networked systems form, evolve, and remain robust. Recent research confirmed that networks formed from large software systems (class collaboration networks in the case of Object-Oriented /OO/ software systems) have a topology common to large real-world and engineered networks and share the already mentioned features.

A large computer program is typically divided into many interacting units at many levels of granularity. In the case of Java programs, the smaller units can be viewed as packages at the first level, classes and interfaces at the second level, and methods at the third level of granularity. Logical connections of classes and interfaces define a network that is often referred to as a *class collaboration network* (from this point on, we do not make the distinction between classes and interfaces – both will be referred to by the term "class"). Classes collaborate in various ways: one class may instantiate and use objects of another class (using objects of different classes as method arguments or return values, and calling methods of objects of different classes); a class can extend the functionality of another class (inheritance), implement some interface presented in the software system, etc. In all cases we say that a class *references* another class. Classes can be viewed as vertices (nodes) and references between them as edges (links or arcs, in case of directional connections) in a class collaboration network, therefore it is natural to represent this kind of network as a directed graph.

Topological measurements of complex networks include degree distribution, the small-world coefficient and the clustering coefficient [1]. Degree distribution $P(k)$, summarizing the connectivity of each node in a network, indicates the probability that a randomly selected node has exactly k incident edges. In the case of directed graphs, node degree can be in-

coming (number of links entering the node), or outgoing (number of links exiting the node), thus there are two types of degree distributions: in-degree and out-degree. While the degree distribution of a random graph (a graph generated by the Erdős and Rényi /ER/ model) is binomial (Poisson in the limit), degree distributions of large real-world networks follow power-law $P(k) \sim k^{-\gamma}$ characteristics [1]. This means that small values of node degree are very common, whereas large values are extremely rare. All networks having this type of distribution are referred to as *scale-free*. Albert and Barabási [1] proposed a one-parameter model for generating scale-free networks with $\gamma = 3$, known as the Barabási-Albert (BA) model. The main properties of this model are network growth (in each iteration one new node is attached to the network), and the rule of preferential attachment (a node with higher degree has higher probability to receive an incoming link from a node that is being attached to the network than a node with lower degree).

Having the small-world property means that the average distance between two randomly chosen nodes in the network is a small value, much smaller than the number of nodes in the network. The distance between two nodes is defined as the number of edges along the shortest path connecting them [1]. The small-world coefficient l, a numerical value that reflects the small-world property of a network, is calculated as the average value of small-world coefficients of each node. The small-world coefficient for a node is the average value of the distance between the selected node and all others directly or indirectly connected to it. Many studies showed that the small-world coefficient of large real-world networks scales proportionally to the natural logarithm of the number of nodes. Networks with $l \sim \ln(\ln(N))$, where N is the number of nodes, are called "ultra small-worlds" [4].

Clustering is a significant characteristic of small-world networks. The inherent tendency to cluster, i.e. the tendency of the neighbors of a node to be neighbors themselves, is quantified by the clustering coefficient. The clustering coefficient is usually computed for undirected graphs, while for networks that have directed links directionality is ignored [12]. The clustering coefficient for node i having k_i edges which connect to k_i other nodes is the ratio between the number of edges that exist between these k_i nodes and the number of edges in the complete graph that consists of k_i nodes, which is $k_i (k_i - 1)/2$. The clustering coefficient of the whole network is the average clustering coefficient of all nodes. In most real-world networks that were analyzed, the clustering coefficient is typically much larger than in a comparable random network (which has the same number of nodes and edges as the real-world network) [1].

1.1. Motivation and Contributions

There exist many studies that focus on questions relating to how software should be written and structured. However, comparatively few studies examine the internal structure of concrete, widely used, software systems. A representative selection of these works, which view software systems as complex networks, is reviewed in Section 2 of this paper.

The initial motivation for this study was to verify the results by Valverde and Solé [18], who showed that the class collaboration network of the Java Development Kit (JDK) has scale-free and small-world properties. In order to achieve the first step of extracting the class collaboration network from JDK, we developed a Java software utility that is able to extract class collaboration networks from arbitrary Java software based on source code analysis. This software utility is described in Section 3.

Next, we computed basic statistical properties of the JDK class collaboration network (gamma exponents of in/out degree distributions, small-world and clustering coefficient) and compared with those reported by Valverde and Solé [18]. However, we extended our research to class collaboration networks associated with several other widely used Java software systems, and analyzed four more class collaboration networks (networks extracted from Ant, Tomcat, Lucene and JavaCC) in order to determine whether they belong to the class of scale-free networks. For each network we detected scale-free or approximately scale-free properties. Small-world and clustering coefficients of these networks were computed and compared with small-world and clustering coefficients of random graphs of the same size. Moreover, for Tomcat and JavaCC networks we detected the ultra small-world property. In addition, we extracted class collaboration networks from ten successive version of one Java software project (Ant from version 1.5.2 to version 1.7.0) and compared them in order to analyze Ant class collaboration network evolution. Section 4 describes these experiments and their results in more detail.

In Section 5 we discuss the implications of the results of our experiments on software engineering. We analyze several theoretical and practical merits of the scale-free properties of networks extracted from software, in various aspects: identification of important classes/interfaces illustrated with examples from the JDK class collaboration network, comments on software testing strategy proposed by Potanin et al. [13] applied to the JDK class collaboration network, connection of the principle of efficient communication among software entities at low cost and the ultra small-world property, and the evolution of the Ant class collaboration network from the perspective of the Barabási-Albert model, where we found that the preferential attachment concept of the BA model can successfully model the evolution of Ant at the structural level.

Finally, in Section 6 we give the conclusion and the possibilities for future work.

2. Related Work

Valverde et al. [17] analyzed the JDK class collaboration network extracted from the class diagram (design level). They found that two largest connected components in the JDK network are scale-free (with gamma exponents 2.5 and 2.65, respectively) and small-world (small-world coefficients are 6.39 and 6.91). They also calculated the clustering coefficient for both components and showed that it is larger than the clustering coefficient predicted by the ER model (clustering coefficients are 0.06 and 0.08). However, in their analysis link directionality was ignored.

The same authors [18] analyzed the JDK class collaboration network that was represented as a directed graph, again extracted from the class diagram, and corrected previous results. They obtained $\gamma^{in} = 2.18$ and $\gamma^{out} = 3.39$ for the largest connected component, and $\gamma^{in} = 2.39$, $\gamma^{out} = 3.3$ for the second largest connected component. Similar values for small-world properties were reported: 5.40 and 5.97 for small-world coefficients, 0.225 and 0.159 for clustering coefficients. They showed that three more Java class collaboration networks formed from UML diagrams (two computer games written in Java, one distributed Java application) are scale-free, small-world, and have clustering coefficients larger than those predicted by the ER model. In order to improve the statistical analysis they also reconstructed (this time from the source code) and analyzed 18 more C/C++ applications. For all analyzed data sets, they obtained in/out degree power-law distributions (γ^{in} in range 1.94 to 2.55, γ^{out} in range 2.41 to 3.39, and for all networks γ^{out} had higher values than γ^{in}), small-world and larger than random graph clustering coefficient properties.

Myers [12] 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 cumulative frequency distribution for in-coming and out-going links and found that these distributions reveal a power-law scaling region. Values of γ^{in} are in range [1.9, 2.5], of γ^{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. [11] investigated collaboration networks of four open source software projects (Linux, XFree86, 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. [13] 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.

Wheeldon and Counsel [19] identified power-law relationships in several OO measures. They analyzed networks that represent different forms of OO coupling (inheritance, aggregation, parameter type and return type). The networks were extracted from three Java software projects (JDK, Ant, Tomcat) using the AutoCode system which parses JavaDoc pages.

Hyland-Wood et al. [4] analyzed software collaboration graphs 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.

Puppin and Silvestry [14] studied the links present among Java classes coming from different contexts (i.e., various unrelated Java software projects). They performed link analysis over the Java class collaboration network that consists of approximately 7700 classes and concluded that the distribution of incoming links follows a power-law curve with $\gamma^{in} = 1$. The same exponent was obtained for the network that contains 49500 classes. Furthermore, they proposed a mechanism for ranking Java classes in this kind of ecosystem, similar to the ranker used in the Google web search engine (PageRank), and presented a prototype search engine for Java classes. Java class collaboration networks analyzed in their work were extracted using data present in JavaDoc documents for related classes.

Finally, scale-free and small-world phenomena were detected in agent-oriented applications [16], grid middleware software [20], inter-package dependency networks in Debian and FreeBSD distributions [7], and software metrics [8]. Also, there is a tendency to incorporate parameters of complex networks into traditional software metrics [9].

3. Extracting Class Collaboration Networks

As part of this research, a Java program called Yaccne has been developed for extracting class collaboration networks from Java source code. Yaccne works in two phases. In the first phase, the nodes of a class collaboration network are formed. Yaccne's source tree crawler passes through all packages in Java software projects, and files with "java" extension in the current package are added into the network as nodes. Inner classes are omitted; they do not exist as nodes in the class collaboration network, but a reference made in the inner class becomes a reference attached to the upper class. In the second phase, links between nodes are determined by the following rules:

1. Class *A* provides an incoming link to class *B* if *A* imports *B*.
2. Class *A* provides an incoming link to class *B* if *B* is in the same package as *A*, and *A* references *B*.
3. In the case of a nondeterministic package import (e.g., import java.io.*) in class *A*, class *A* provides incoming links just to classes that it references, not to all classes from the imported package.
4. Class *A* provides an incoming link to class *B* if *A* references *B* through its full package path (e.g., suppose *A* contains statement new java.io.IOException("Error"), and IOException is not imported through a Java import statement. Then class *A* references class IOException from package java.io).
5. References that come from outside of the software system are excluded, thus we treat the software project and analyze appropriate class collaboration networks as an isolated system.

In order to perform node linkage, the scanner and the parser for the Java 1.5 language are built into Yaccne. The scanner and the parser are realized using the JavaCC [5] compiler generator. We modified the Java grammar that comes with this software to acquire the following information: list of imported classes (data used in rule 1), list of imported packages (rule 3), list of referential data types (rule 2), list of referential data types that contain dots in the name (rule 4) and some statistical/general information (number of fields, number of methods, is the class abstract, is the parsed entity an interface, which class extends the parsed class). Table 1 shows a syntactically correct Java class and the result of parsing described above.

Table 1. Example of a syntactically correct Java class (left) and the result of Yaccne parsing (right)

```
import java.util.LinkedList;
import java.io.*;
import java.awt.*;
import java.math.BigInteger;

public abstract class Simple extends Simpler {
   private LinkedList<BigInteger> lbi =
       new LinkedList<BigInteger>();

   public void calc() {
     C c = new C();  D d = new D();
     ee.ff.Merge merger = new
                    ee.ff.AdvancedMerge(c, d);
     merger.converToBigI(lbi);
   }
   public abstract void advancedCalc();
}
```

```
Imported classes:  java.util.LinkedList,
java.math.BigInteger

Imported packages:  java.io.*, java.awt.*

Referential types: D, Simpler, C, LinkedList,
BigInteger

Referential types containing dot:
ee.ff.AdvancedMerge,
ee.ff.Merge

Interface: false
Abstract class: true
Simple.java extends: Simpler
Number of fields: 1
Number of methods: 2
```

Yaccne uses Jung-1.7.4 [6], a Java library that provides a variety of graph algorithms. The small-world coefficient per node, diameter of the network (size of the longest path in the network), clustering coefficient (link direction is ignored) and PageRank per node are computed using this library. Yaccne generates the following files as output:

- stats.dat: general statistics of the analyzed project (number of nodes, number of links, total number of lines of code, number of interfaces, number of abstract classes, number of fields, number of methods, network diameter, small-world coefficient and clustering coefficient),
- nodes.dat: information about each node in the network (node name, in-degree, out-degree, number of lines of code, number of fields, number of methods, name of the base class or empty if class does not extend any class, depth in inheritance tree, is the node an interface, is the node an abstract class, small-world coefficient, clustering coefficient, and node PageRank),
- links.dat: information about each link in the network (source node name, destination node name and description that says by which rule the link is formed),
- in.distr: unnormalized cumulative in-degree frequency distribution,
- out.distr: unnormalized cumulative out-degree frequency distribution.

Yaccne contains a subprogram called Diff that finds differences between two networks where the second network evolves from the first one by adding new nodes, deleting some nodes, adding new links and/or deleting some links. This program takes as input nodes.dat and links.dat of the first and the second network and generates two reports: one that contains node differences and one that contains link differences. With the first report we can examine the properties of new nodes (degree and PageRank), and with the second we can test the preferential attachment hypothesis (determine to which "old" nodes new links are attached).

4. Experiments and Results

Class collaboration networks from five large Java software projects (JDK, Tomcat 6.0.13, Ant 1.7.0, Lucene 2.1.0 and JavaCC 3.2.0) have been extracted and analyzed. After that ten class collaboration networks from Ant version 1.5.2 to version 1.7.0 have been extracted and compared.

The main statistical characteristics of analyzed class collaboration networks have been computed and summarized in Table 2.

Table 2. Overall statistics for analyzed software systems

	JDK	Tomcat	Ant	Lucene	JavaCC
Number of nodes	1878	1046	778	354	79
Number of interfaces	381	139	65	17	2
Number of abstract classes	241	65	59	29	2
Number of links	12806	4646	3634	2221	274
Number of nodes without incoming links	501	264	261	126	26
Number of nodes without outgoing links	136	108	86	49	13
Number of nodes without in- and out-links	30	50	15	9	2
Max. incoming links	284	193	459	110	22
Max. outgoing links	93	61	34	30	29
Diameter	21	11	15	10	6
Max. inheritance depth	6	3	6	4	2
Total lines of code	245459	154814	93510	40904	13612

4.1. Degree Distributions

As a result of our experiments, unnormalized cumulative frequency distributions $N_c^{in}(k)$ and $N_c^{out}(k)$ for each extracted class collaboration network have been computed, where $N_c(k)$ indicates the number of nodes in a network with degree greater than or equal to k. $N_c(k)$ is an unnormalized integral of the probability distribution $P(k)$, and if $P(k) \sim k^{-\gamma}$, then $N_c(k) \sim k^{-\gamma+1}$ (the indefinite integral of a power function is the power function with the exponent increased by one). A linear function was fitted to the linear ranges of log-log plotted distributions in order to estimate the value of the gamma exponent. Figures 1, 2, 3, 4 and 5 show in-coming and out-going degree distributions for JDK, Ant, Tomcat, Lucene and JavaCC, respectively (with values of the Pearson correlation coefficient R, and standard deviation SD), and Table 3 lists gamma exponent values with the square of the Pearson correlation coefficient (R^2) for the linear fit. Gamma exponent values for in-coming and out-going link frequency distribution for JDK are similar to those reported by Valverde and Solé [18]. This result is significant because we used different sources for extracting the class collaboration network – we have done it from the source code, Valverde and Solé from the class diagram. Linear ranges of the distributions are followed by a faster decay at the large number of connections (k). This phenomenon was also reported by Myers [12] and Hyland-Wood et al. [4]. The Pearson correlation coefficient gives us the quality of the linear fit. If 0.95 is taken as a minimal reliable value, we can state a power law for the JDK in-coming, JDK out-going, Ant in-coming, Tomcat out-going (very close to 0.95) and JavaCC out-going link degree distributions. Other distributions can be considered as approximately power-law (R^2 values are higher than 0.90, except for JavaCC where R^2 is very close to 0.90). For all analyzed networks we determined that $\gamma^{out} \geq \gamma^{in}$, except for JavaCC.

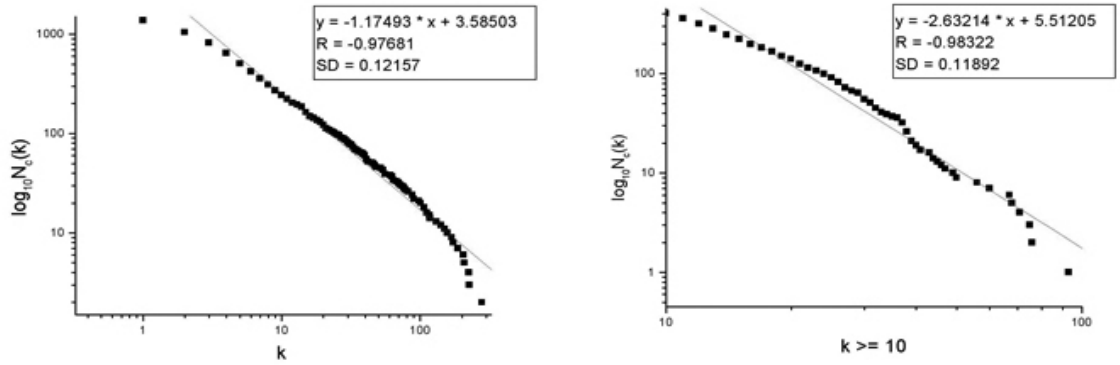

Figure 1. Unnormalized in-coming (left) and out-going (right) link frequency distributions for the JDK class collaboration network. For the out-going link distribution we fitted a linear function for degrees greater than or equal to 10

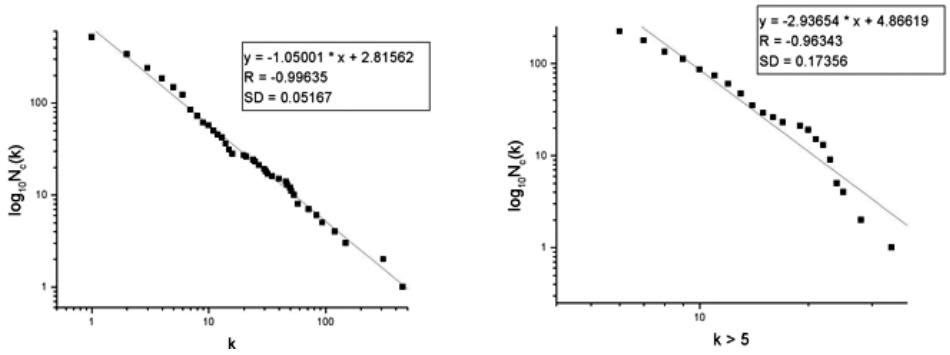

Figure 2. Unnormalized in-coming (left) and out-going (right) link frequency distributions for the Ant class collaboration network. For the out-going link distribution we fitted a linear function for degrees greater than 5

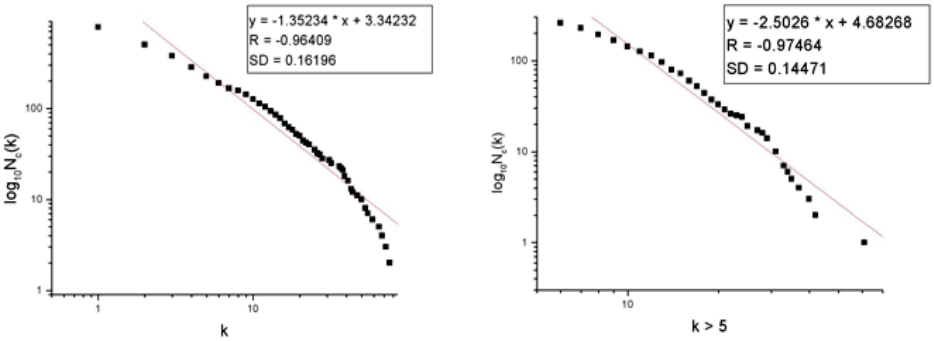

Figure 3. Unnormalized in-coming (left) and out-going (right) link frequency distributions for the Tomcat class collaboration network. For the out-going link distribution we fitted a linear function for degrees greater than 5

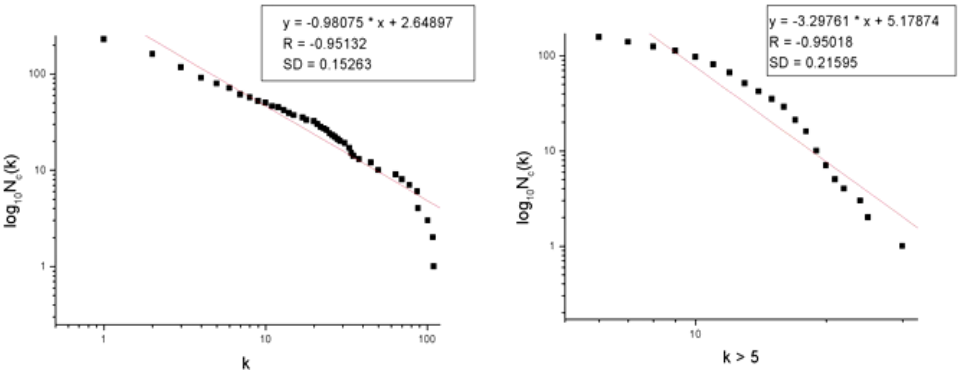

Figure 4. Unnormalized in-coming (left) and out-going (right) link frequency distributions for the Lucene class collaboration network. For the out-going link distribution we fitted a linear function for degrees greater than 5

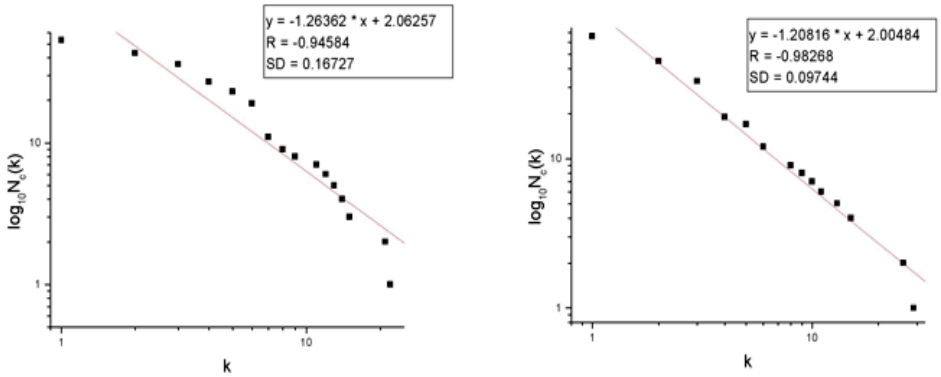

Figure 5. Unnormalized in-coming (left) and out-going (right) link frequency distributions for the JavaCC class collaboration network

Table 3. Gamma exponents with the square of the Pearson correlation coefficient for linear fit per class collaboration network

Class collaboration network	γ^{in}	R^2_{in}	γ^{out}	R^2_{out}
JDK	2.17493	0.9541	3.63214	0.9667
Ant	2.05001	0.9927	3.93654	0.9281
Tomcat	2.35234	0.9294	3.5026	0.9499
Lucene	1.98075	0.9050	4.29761	0.9028
JavaCC	2.26362	0.8946	2.20816	0.9656

The fact that degree distributions presented in Figures 1–5 can be well approximated by power laws tells us that the node-degree characteristic of class collaboration networks is highly variable: there is a broad spectrum of values it can take and statistically significant numbers of nodes with degree which is far from the average node degree. This degree heterogeneity property is discussed more deeply in Section 5.

4.2. Small-World and Clustering Coefficient

Let us denote the average number of links per node with μ. In all analyzed networks μ is much smaller than the total number of nodes N, which indicates that the networks are sparse [11]. The small-world property is very meaningful in sparse networks because, despite large network size, there exists a relatively short path between any pair of nodes.

A random network with given N and μ ($N >> \mu$) is characterized by having small values of l (small-world coefficient) and c (clustering coefficient). For random networks the small-world coefficient can be approximated by $l_{rand} \sim \ln(N)/\ln(\mu)$ and the clustering coefficient by $c_{rand} \sim \mu/N$ [11].

Table 4 shows small-world and clustering coefficients for the analyzed networks, with values of small-world and clustering coefficients for random graphs of the same size. Small-world and clustering coefficient values for the JDK class collaboration network are close to those reported by Valverde and Solé [18]. As pointed out in Section 2, the class collaboration networks explored in [17], [18] and [11] have small-world coefficients approximately equal to the small-world coefficients of random graphs of the same size, and higher clustering coefficients. The same holds for the class collaboration networks we analyzed, except for the small-world values of the Ant and JavaCC class collaboration networks. These two networks can be considered as ultra small-worlds ($l_{Tomcat} \sim \ln(\ln(N_{Tomcat}))$ and $l_{JavaCC} \sim \ln(\ln(N_{JavaCC}))$. In all networks, $c >> c_{rand}$.

Table 4. Small-world and clustering coefficient values of analyzed networks with expected values for random graphs of the same size

	#nodes	#links	$\mu/2$	l	l_{rand}	c	c_{rand}
JDK	1878	12806	6.82	4.391	3.93	0.453	0.0036
Ant	778	3634	4.67	4.131	4.32	0.505	0.0060
Tomcat	1046	4646	4.44	1.909	4.66	0.464	0.0042
Lucene	354	2221	6.27	2.278	3.19	0.386	0.0177
JavaCC	79	274	3.47	1.220	3.52	0.437	0.0439

4.3. Evolution of the Ant Class Collaboration Network

Ten class collaboration networks extracted from successive versions of Ant (versions 1.5.2, 1.5.3, 1.5.4, 1.6.0, 1.6.1, 1.6.2, 1.6.3, 1.6.4, 1.6.5 and 1.7.0) were analyzed and changes from one version to the next version were observed. From the theoretical perspective, an artificial complex network generated by the BA model, or its modifications which employ the preferential attachment principle, evolves into a perfect scale-free state. Ant is the ideal candidate, among the software systems studied in this paper, to test whether the preferential attachment principle for newly added nodes can explain scale-free state of some real-world software network. Ant's class collaboration network evolves from version 1.5.2 to version 1.7.0 into almost perfect scale-free state: a power law nearly ideally fits Ant's complementary cumulative incoming degree distribution in version 1.7.0 (see Figure 2, left).

From version 1.5.2 to version 1.5.4 modifications in the class collaboration network structure were minor. All three networks have the same nodes (there were no new classes or interfaces added, and nothing was removed from the project), but the 1.5.3 network has three new links with respect to 1.5.2, and 1.5.4 has one more link than 1.5.3. We detected the first massive change in class collaboration network structure in the transition from version 1.5.4 to version 1.6.0. In 1.6.0 there exist 114 new nodes which did not exist in the 1.5.4 network. Ten of them are new interfaces and 104 are new classes. All nodes from the 1.5.4 network remain in the 1.6.0 network (nothing was deleted). In Figure 6 (left) we plot the series of pairs (x_k, y_k) for each node in the 1.5.4 network, where x_k represents the number of in-coming links for node k in network

1.5.4 and y_k the number of newly attached in-coming links by nodes that are added as new in the 1.6.0 network (which do not exist in the 1.5.4 network). This chart shows that nodes which had higher in-coming link degree in the 1.5.4 network received more in-coming links from nodes added in version 1.6.0 than those with smaller in-coming link degrees. The node org.apache.tools.ant.BuildException, which has 336 in-coming links in the 1.5.4 network (the node with the highest in-coming link degree) received 63 new in-coming links by newly introduced nodes (which is the highest number of newly added in-coming links to one node). Similar behavior was observed at the node with the second largest in-degree (org.apache.tools.ant.Project – in the 1.5.4 network it has 220 in-coming links, receiving 43 links from nodes which appeared in the 1.6.0 network), and with the third largest as well (org.apache.tools.ant.Task – in the 1.5.4 network it has 124 in-coming links). These three nodes are the three most preferential nodes viewed from the perspective of the Barabási-Albert model (having the highest in-coming link degree) and they received the most in-coming links from nodes added in the new version of the software.

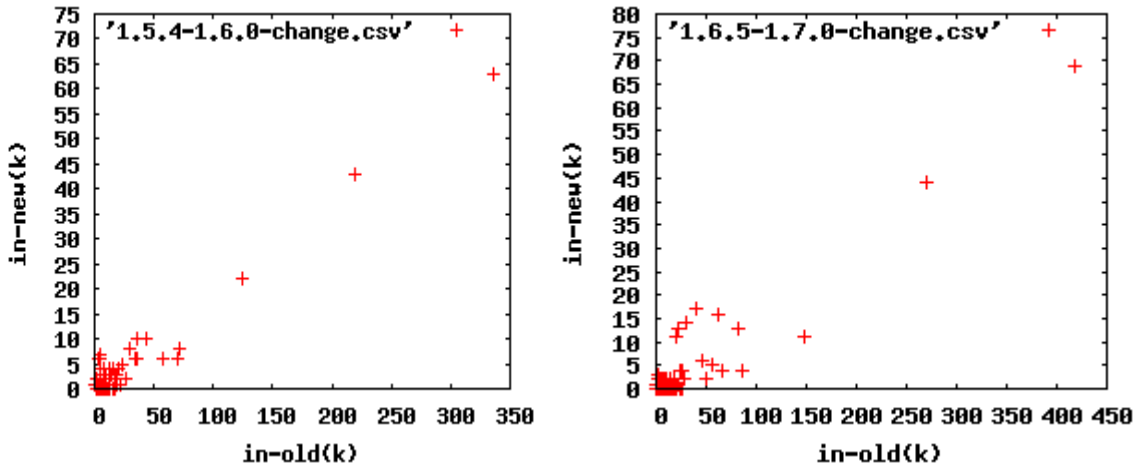

Figure 6. Left: Plot of series of pairs (x_k, y_k) for each node in the Ant 1.5.4 network, where x_k represents the number of in-coming links for node k in network 1.5.4 and y_k the number of newly attached in-coming links by nodes that were added as new in the 1.6.0 network. Right: Analogous plot for the version transition from 1.6.5 to 1.7.0

Changes in the structure of the class collaboration network from version 1.6.0 to 1.6.1 were not drastic. Four nodes and 18 links were added to the network. All nodes in 1.6.0 appear also in 1.6.1 (nothing was deleted). Seventeen links out of 18 new ones are attached as in-coming links to 1.6.0 nodes. Most of them are distributed to the preferential nodes (BuildException, Project /the two nodes with highest in-degree/, org.apache.tools.ant.types.Commandline and org.apache.tools.ant.taskdefs.Execute).

Version 1.6.2 brings 16 new nodes and 75 new links, but only 27 links (out of 75) are links from new nodes to the old ones. Most of them again point to the preferential nodes: BuildException (7), Project (4) and Task (4).

Version 1.6.3 brings 20 new nodes and 141 new links. Eighty-seven (out of 141) are interesting – the ones that point to the old nodes from the new nodes. The largest numbers of in-coming links were assigned to classes that had the largest number of in-coming links before adding new nodes (BuildException, 17 new in-coming links). Thirty-six links were attached to the 16 nodes that have an in-coming link degree higher than ten, 21 to the 20 nodes that have incoming link degree higher than or equal to 5, and 13 to the 7 nodes that have an in-coming link degree less than 5.

The class collaboration network extracted from version 1.6.4 has minor differences to the extracted collaboration network from the previous version (1.6.3): both have the same nodes, two new links are present in the 1.6.4 network.

Version 1.7.0 brings big structural changes in the class collaboration network. There are 132 new nodes, and for the first time some nodes from the previous collaboration network were deleted (44 of them). The total number of new links is 634, 129 pointing from some new node to some new node, 144 from some old node to another old node, and 361 from some new node to some old node. Figure 6 (right) plots the series of pairs (x_k, y_k) for each node in the 1.6.5 network, where x_k represents the number of in-coming links for node k in network 1.6.5, and y_k the number of newly attached in-coming links from nodes that are added as new in the 1.7.0 network. As in the case of the 1.5.4 to 1.6.0 version transition (depicted in Figure 6, left), in Figure 6 (right) we observe that the two nodes with the highest in-coming link degree (BuildException and Project) received the most in-coming links.

The same type of real-world network analysis presented here can be performed on other software systems as well, and possible evidence of preferential attachment would suggest that this principle is tolerant to small fluctuations from perfect scale-free state in class collaboration network structure.

5. Implications on Software Engineering

In the previous section we showed that extracted class collaboration networks from selected Java software projects are scale-free, or approximately scale-free. This result has both theoretical and practical values. First of all, it suggests that similar organization principles are the foundation for the evolution of software systems and other complex networks, such as social and biological networks, which also have the scale-free property [1]. This implies that the study of complex networks, in general, can be a useful heuristic to understand structural complexity and evolution of software systems.

Based on the scale-free property of the class collaboration network, important classes/interfaces in a large Java software system can be identified. The most prominent feature of a scale-free network is the power-law degree distribution. The direct effect of a power-law distribution in Java class collaboration networks is that there are a few classes/interfaces with a much larger number of connections (high in-coming or out-going node degree) to other classes/interfaces compared to the average. As noted by Myers [12], a high in-degree typically results from code reuse. It is very difficult to change highly reused classes/interfaces because of their importance for the stability of the system [18]. A high number of outgoing links indicates that a class has a high degree of aggregation, and nodes with small out-degree are generally simple, since they do not aggregate other classes [12].

In our study we identified top ten highest in-degree and out-degree classes/interfaces in JDK. They are shown in Tables 5 and 6, respectively. As can be seen, classes java.awt.Component and javax.swing.JComponent are in the top ten classes by both in-degree and out-degree, which means that these classes exhibit a high degree of reuse and aggregation at the same time. These two classes have both significant internal complexity (due to aggregating the behavior of several other classes) and significant external responsibility (because they are used in a lot of other classes in the system).

Table 5. Top ten highest in-degree nodes in JDK

Class name	#InLinks	#OutLinks	PageRank	LOC	C_p
java.io.IOException	284	0	0.0416	10	0
java.io.Serializable	279	0	0.484	3	0
java.awt.Component	226	76	0.0087	2489	734292275264
java.awt.Graphics	225	11	0.0055	163	998476875
java.awt.Rectangle	208	5	0.0064	243	262828800
javax.swing.JComponent	205	68	0.004	2045	397391762000
java.awt.Dimension	187	2	0.0051	56	7833056
java.util.Vector	174	7	0.0044	332	477694728
javax.swing.plaf.ComponentUI	169	7	0.0051	45	62977005
java.awt.Color	157	15	0.0029	452	2506803300

Table 6. Top ten highest out-degree nodes in JDK

Class name	#OutLinks	#InLinks	PageRank	LOC	C_p
java.awt.Toolkit	93	25	0.002	595	3216346875
java.awt.Component	76	226	0.0087	2489	734292275264
javax.swing.JTable	75	7	0.0002	2549	702568125
javax.swing.text.JtextComponent	71	40	0.0006	1498	12082268800
javax.swing.JComponent	68	205	0.004	2045	397391762000
javax.swing.text.html.HTMLEditorKit	67	6	0.0001	972	157079088
javax.swing.plaf.basic.BasicTreeUI	60	3	0.0001	2324	75297600
javax.swing.JEditorPane	56	10	0.0001	1047	328339200
javax.swing.JTree	50	8	0.0003	2208	353280000
javax.swing.AbstractButton	49	33	0.0007	1145	2993818905

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 [3]: *fan-in* (the number of other functions calling a given function in a module) and *fan-out* (the number of other functions being called from a given function in a module). Actually, in-degree and out-degree are fan-in and fan-out analogues at the class collaboration level. Henry and Kafura [3] defined a complexity metric C_p calculated as LOC · (*fan-in* · *fan-out*)^2, where LOC represents the number of lines of code in a software entity. According to Zimmermann et al. [21], components with a large fan-in and fan-out may indicate poor design and such modules have to be decomposed

correctly. If C_p for the class is defined as LOC · (*in-degree* · *out-degree*)^2 then, in the JDK case, two classes with the highest complexity are java.awt.Component and javax.swing.JComponent, and their C_p is at least one order of magnitude higher than others (C_p values for the top ten in-degree/out-degree nodes are shown in Tables 5 and 6).

One important aspect of scale-free networks is their robustness to damage [1]. In the case of software systems with scale-free topology at the class collaboration level this means that random errors may not cause major crashes of the whole system. This can explain why many large software systems can function properly most of the time even with software defects. However, if errors happen within classes that have high in-coming degree (classes with significant external responsibility), this can cause the whole system to fail. On this basis Potanin et al. [13] suggested that by concentrating debugging methodologies on such well-connected classes, rather than the small ones, software engineers may be able to improve the reliability of code more efficiently. The idea is simple: first eliminate bugs from the classes that have major impact on the system stability, then deal with other classes. The first step of this debugging approach is to identify classes that have highest in-coming degree. Therefore, in the JDK case, classes from the top ten list will be tested first. We can see that in the top ten list we have classes/interfaces with a high number of in-coming links and small number of out-going links (for example, java.io.IOException is the most reused class and java.io.Serializable the most implemented interface, both entities are without out-going links – no aggregation, having 10 and 3 lines of code, respectively, $C_p = 0$ for both entities). These entities are quite simple and do not use other components, so with a little effort it can be determined if they are functioning properly. On the other side, we can see in the top ten list the classes with high in-degree and high out-degree at the same time (already mentioned java.awt.Component and javax.swing.JComponent). In order to validate these classes, all classes used by them must be previously validated. Obviously, classes with higher values of the C_p are "problematic" (in the sense of having higher priority) in the debugging strategy described above.

One of the recognized principles to follow in software development is the principle of efficient communication among software entities at low cost, that is, a software system should have a relatively average shortest path length [10]. In other words, transport of information between classes is more efficient when the small-world coefficient of the class collaboration network is smaller. In this research we showed that all analyzed class-collaboration networks have the small-world property. It is interesting that we found two networks with the ultra small-world property: Tomcat and JavaCC. Cohen and Havlin [2] showed that the lower bound on the small-world coefficient of any scale-free network with gamma exponent greater than 2 is of the order of ln(ln(N)) (N is the number of nodes in the network). Class collaboration networks of Tomcat and JavaCC can be considered as approximately scale-free and both have gamma-in/gamma-out exponents greater than 2 (see Table 3). This means that this principle of efficient communication at low cost is maximized in those Java projects.

In Section 4.3 we described the evolution of the Ant class collaboration network extracted from version 1.5.2 to version 1.7.0. Our motivation for this analysis was to check the preferential attachment hypothesis introduced by the Barabási-Albert model, in order to determine whether that hypothesis can model the evolution of the Ant software system at the structural level. The BA model states two conditions for a scale-free topology. The first is that the network is growing. If, for example, we observe the 1.5.4 to 1.6.0 network transition, we will see that 104 nodes were added to network 1.5.4 which results in network 1.6.0, therefore this first condition of the BA model is satisfied. The second condition of the BA model is that nodes with a higher in-coming link degree have a higher probability of receiving an in-coming link from newly introduced nodes than nodes with a lower in-coming link degree. In the Ant class collaboration network's transition from version 1.5.4 to version 1.6.0 we have the situation that the highest in-coming link degree nodes (classes org.apache.tools.ant.BuildException, org.apache.tools.ant.Project and org.apache.tools.ant.Task) received the most in-coming links by newly added nodes, thus the second condition of the BA model is also satisfied. Similar behavior can be detected in the transitions 1.6.0–1.6.1, 1.6.1–1.6.2, 1.6.2–1.6.3 and 1.6.5–1.7.0. We can conclude that the Barabási-Albert concept of preferential attachment can successfully model the structural changes in the Ant class collaboration network.

However, the BA model has two disadvantages in order to be a good predictive model for software evolution at the structural level. First, it generates scale-free topology with gamma exponent that has a constant value 3. Second, it cannot produce hierarchical structures [12]. This is very important because it was observed that engineered design leads to hierarchical structures [15]. Our future work will be focused on extending the BA model (making a new model on the basis of the preferential attachment hypothesis) in order to facilitate these two capabilities.

6. Conclusion and Future Work

This paper investigated statistical properties (degree distributions, small-world and clustering coefficients) of class collaboration networks formed in five large Java software projects (JDK, Ant, Tomcat, Lucene and JavaCC) and showed that these networks exhibited scale-free or nearly scale-free and small-world properties, while for Tomcat and JavaCC we detected the ultra small-world property. Furthermore, we showed that the clustering coefficient value of all

examined networks is significantly larger than the clustering coefficient value for the random graph of the same size. Pure topological properties and phenomena that were detected enabled us to derive implications related to practical software engineering issues in several aspects: identification of classes important to software stability and evolution; Henry-Kafura metrics and their relation to problems with the software testing strategy proposed in [13]; detection of maximal utilization of the communication efficiency principle. The first two aspects were discussed with regards to the JDK class collaboration network, and the last one from the perspective of class collaboration networks with scale-free and ultra small-world properties.

In order to check the preferential attachment rule of the BA model, we captured class collaboration networks from ten successive versions of Ant, and analyzed changes in network structure. Results show that nodes with higher in-coming link degree receive more in-coming links from new nodes than others. This means that the preferential attachment concept introduced in the BA model can explain structural changes in the Ant's class collaboration network and provide clues about how the network evolved into a scale-free state (the state that was identified by observed degree distributions that follow the power law). On the other hand, the BA model is incapable of generating hierarchical structures, which is clearly relevant for software design. Our future work will include an investigation of class collaboration network evolution in other large Java software projects with more evolutionary steps, in order to obtain further empirical evidence of software evolution at collaboration (structural) level, and defining a model that incorporates the preferential attachment concept with the capability of generating hierarchical structures.

Acknowledgments

The authors gratefully acknowledge the support of this work by the Serbian Ministry of Science and Technological Development through project *Intelligent Techniques and Their Integration into Wide-Spectrum Decision Support*, no. OI174023. The authors would also like to thank the anonymous reviewers for their valuable comments.

References

[1] **R. Albert, A.-L. Barabási.** Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74, 2002, 47–97.

[2] **R. Cohen, S. Havlin.** Scale-free networks are ultra-small. *Phys. Rev. Lett.*, 90, 2003, 058701.

[3] **S.M. Henry, D.G. Kafura.** Software structure metrics based on information flow. *IEEE T. Software Eng.*, 7, 1981, 510–518.

[4] **D. Hyland-Wood, D. Carrington, S. Kaplan.** Scale-free nature of Java software package, class and method collaboration graphs. *Technical report no.TR-MS*1286, *MIND Laboratory, University of Maryland College Park*, 2006.

[5] JavaCC. *javacc: JavaCC Home*, 2010. *https://javacc.dev.java.net/*.

[6] Jung. *JUNG: Java Universal Network/Graph Framework*, 2010. *http://jung.sourceforge.net/*.

[7] **N. La Belle, E. Wallingford.** Inter-package dependency networks in open-source software. *Proc.* 6*th Int. Conf. on Complex Systems* (*ICCS*), 2006, *paper no.* 226.

[8] **J. Liu, K. He, Y. Ma, R. Peng.** Scale free in software metrics. *Proc.* 30*th Annual Int. Computer Software and Applications Conf.* (*COMPSAC*), 2006, 229–235.

[9] **Y. Ma, K. He, D. Du, J. Liu, Y. Yan.** A complexity metrics set for large-scale object-oriented software systems. *Proc.* 6*th IEEE Int. Conf. on Computer and Information Technology* (*CIT*), 2006, 189–194.

[10] **Y. Ma, K. He, D. Du, J. Liu.** Network motifs in object-oriented software systems. *Dyn. Contin. Discret. Impuls. Syst. Ser. B: Appl. Algorithms*, 14(S6), 2007, 166–172.

[11] **A.P. de Moura, Y.C. Lay, A.E. Motter.** Signatures of small-world and scale-free properties in large computer programs. *Phys. Rev. E*, 68, 2003, 017102.

[12] **C.-R. Myers.** Software systems as complex networks: Structure, function and evolvability of software collaboration graphs. *Phys. Rev. E*, 68, 2003, 046116.

[13] **A. Potanin, J. Noble, M. Frean, R. Biddle.** Scale-free geometry in object-oriented programs. *Commun. ACM*, 48(5), 2005, 99–103.

[14] **D. Puppin, F. Silvestri.** The social network of Java classes. *Proc.* 2006 *ACM Symposium on Applied computing (SAC)*, 2006, 1409–1413.

[15] **R.V. Solé, R. Ferrer Cancho, J.M. Montoya, S. Valverde.** Selection, tinkering, and emergence in complex networks, *Complexity*, 8, 2003, 20–33.

[16] **J. Sudeikat, W. Renz.** On complex networks in software: How agent-orientation effects software structures. *Proc.* 5*th Int. Central and Eastern European Conf. on Multi-Agent Systems* (*CEEMAS*), *Lect. Notes Comput. Sc.*, 4696, 2007, 215–224.

[17] **S. Valverde, R. Ferrer Cancho, R.V. Solé.** Scale-free networks from optimal design. *Europhys. Lett.*, 60, 2002, 512–517.

[18] **S. Valverde, R.V. Solé.** Hierarchical small-worlds in software architecture. *Dyn. Contin. Discret. Impuls. Syst. Ser. B: Appl. Algorithms*, 14(S6), 2007, 1–11.

[19] **R. Wheeldon, S. Counsell.** Power law distributions in class relationships. *Proc.* 3*rd IEEE Int. Workshop on Source Code Analysis and Manipulation* (*SCAM*), 2003, 45–54.

[20] **P. Yuan, H. Jin, K. Deng, Q. Chen.** Analyzing software component graphs of grid middleware: Hint to performance improvement. *Proc.* 8*th Int. Conf. on Algorithms and Architectures for Parallel Processing* (*ICA3PP*), *Lect. Notes Comput. Sc.*, 5022, 2008, 305–315.

[21] **T. Zimmermann, N. Nagappan, A. Zeller.** Predicting bugs from history. *In T. Mens, S. Demeyer (Eds.), Software Evolution, Springer*, 2008, 69–88.

Received July 2010.