Computing (2017) 99:1055–1079
DOI 10.1007/s00607-017-0549-6

Analysis of high structural class coupling in object-oriented software systems

Miloš Savić[1] · Mirjana Ivanović[1] · Miloš Radovanović[1]

Received: 5 May 2016 / Accepted: 1 March 2017 / Published online: 10 March 2017

Abstract Understanding coupling between classes in object-oriented (OO) software systems is useful for a variety of software development and maintenance activities. In this paper we propose a novel, network-based methodology to analyze high structural class coupling in OO software systems. The proposed methodology is based on statistically robust structural analysis of class collaboration networks whose nodes are enriched with both software metrics and domain-independent metrics used in analysis of complex networks. To demonstrate the usefulness of the methodology we analyze five open-source, large-scale software systems written in Java. Contrary to frequently reported findings, the obtained results indicate that high structural class coupling in real software systems cannot be accurately modeled by power-law distributions. Our analysis also shows that highly-coupled classes tend to be significantly more voluminous and functionally important compared to loosely coupled classes, and do not tend to be localized in class inheritance hierarchies. Finally, in four out of five analyzed systems highly coupled classes tend to have drastically higher afferent than efferent coupling. This implies that the existence of high class coupling in an OO software system would rather indicate negative aspects of extensive internal class reuse than negative aspects of extensive internal class aggregation.

✉ Miloš Savić
svc@dmi.uns.ac.rs

Mirjana Ivanović
mira@dmi.uns.ac.rs

Miloš Radovanović
radacha@dmi.uns.ac.rs

1 Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia

Keywords High coupling · Class collaboration networks · Network analysis · Software metrics · Power-law · Class reuse · Class aggregation

Mathematics Subject Classification 05C82 Small world graphs, complex networks · 68N30 Mathematical aspects of software engineering (specification, verification, metrics, requirements, etc.)

1 Introduction

"Low coupling, high cohesion" is one of basic design principles in software engineering [30]. This principle states that the coupling between modules of a software system has to be as minimal as possible, at the same time keeping strong relations between constituent elements of each module. The complete class coupling structure of an OO software system can be represented by a directed graph that is also known as a class collaboration graph [20], class dependency network [25] or class collaboration network [24]. Namely, nodes of class collaboration networks represent individual classes, while two classes A and B are connected by the directed link $A \rightarrow B$ if class A references class B.

Internal class reuse (afferent coupling) and internal class aggregation (efferent coupling) jointly constitute the structural coupling of a class. Internal class reuse can be considered as good software engineering practice, since it reduces or eliminates duplicated code inside the system. However, there are two potential negative aspects of extensive class reuse: high criticality and low testability of highly reused classes [5]. On the other hand, high internal class aggregation can negatively impact the following external attributes of software systems: understandability, error-proneness, and external reusability [5].

Classes that are highly coupled are also called *hubs*. In recent years, researchers investigated a variety of real-world software systems under the framework of complex network theory, revealing the presence of the scale-free property [2] in class collaboration networks, as well as networks representing software systems at other levels of abstraction. A network exhibits the scale-free property if its degree distribution (the distribution of the number of links incident to a node) can be described by a power-law, i.e. a relation of the form $Y \sim X^{-\gamma}$ where Y denotes the fraction of nodes having X incident links and γ is the power-law scaling exponent. The scale-free property of a class collaboration network directly implies the existence of hubs in the corresponding software system. Additionally, the scaling exponent of the fitted power-law can be used as a metric of software design quality. However, empirically observed degree distributions of class collaboration networks in the majority of previous studies were tested only against a power-law, and often by applying linear regression on log-log plots which is a biased methodology to confirm the presence of power-laws and accurately estimate their scaling exponents [8]. This further implies that the current state of empirical evidence does not provide a clear indication weather class collaboration networks are truly scale-free, and, consequently, weather the power-law scaling exponent can be reliably used as a software metric. Additionally, previous studies have not addressed characteristics of afferent and efferent coupling of hubs, as well as their distinctive characteristics compared to loosely coupled classes.

In this paper we propose a novel, network-based methodology to analyze high structural class coupling in real-world software systems. The methodology encompasses statistically robust techniques to investigate the structure of *enriched* class collaboration networks—class collaboration networks whose nodes are enriched with metric vectors containing both software metrics and domain-independent metrics used in the analysis of complex networks. The metric vectors attached to nodes quantify internal complexity, inheritance and functional importance of classes present in the system and enable us to determine characteristics of highly coupled classes with respect to those attributes. Our methodology encompasses the following techniques:

- Degree distribution analysis based on a robust statistical test introduced by Clauset et al. [8] in order to examine properties of empirically observed degree distributions. To the best of our knowledge, this is the first time that this robust statistical procedure is considered to analyze degree distributions of class collaboration networks.
- The identification of distinctive characteristics of hubs using a metric-based comparison test proposed in this paper. The metric-based comparison test combines the Mann–Whitney test [18] and probabilities of superiority [11] applied to metric vectors attached to nodes of enriched class collaboration networks. The main goal of the test is to determine key differences between hubs and loosely coupled classes regarding their internal complexity, inheritance and functional importance. Knowing these differences may lead to a deeper understanding of high class coupling present in a concrete software system, which in turn can be useful for planning and estimating impacts of software maintenance activities.
- The analysis of the balance between afferent and efferent coupling of hubs in order to detect whether high class coupling was dominantly caused by extensive class reuse, by extensive class aggregation, or significantly by both factors. The main goal of this analysis is to determine what high class coupling in a concrete software system may actually indicate—the negative aspects of extensive class reuse, the negative aspects of extensive class aggregation or the negative aspects of both.

To demonstrate the usefulness of the methodology we applied it to five enriched class collaboration networks extracted from the source code of five large-scale software systems written in Java.

The rest of the paper is structured as follows. Necessary preliminaries and definitions are given in Sect. 2. Section 3 presents related works. Our methodology to analyze high class coupling is described in Sect. 4. The next section, Sect. 5, presents the application of our methodology to real-world software systems. The importance of obtained results for software engineering practitioners is discussed in Sect. 6. The last section presents conclusions and directions for future work.

2 Preliminaries

Class collaboration networks describe structural dependencies between classes and interfaces defined in object-oriented software systems. They can be more formally defined as follows.

Definition 1 (*Class collaboration network*) A class collaboration network corresponding to an object-oriented software system S is the directed graph whose nodes represent classes and interfaces of S. Two nodes A and B are connected by the directed link $A \rightarrow B$ if any of the following holds:

- A extends B or implements B (in the case that B is an interface),
- A declares a member variable (field, class attribute) whose type is B,
- A instantiates objects whose type is B by calling a constructor of B,
- A contains a method that declares a local variable whose type is B,
- A contains a method that has a parameter whose type is B,
- A contains a method whose return type is B,
- A contains a method that access a member variable declared in B, or
- A contains a method that calls a method declared in B.

If $A \rightarrow B$ then we also say that (1) classes A and B are coupled, (2) class A internally aggregates class B, and (3) class B is internally used by class A. If there is another class C defined in the system such that $C \rightarrow B$ then we say that class B is internally reused since it is used by more than one class.

Definition 2 (*Node degree*) The degree of node i in network G, denoted by k_i, is the number of links incident with i. If G does not contain parallel links (different links connecting the same pair of nodes) then k_i is equal to the number of nodes to which i is directly connected.

Definition 3 (*Degree distribution*) The degree distribution of network G is given by the probability mass function $P(k) = P\{D = k\}$, where D is a random variable that represents the degree of a randomly chosen node. In other words, $P(k)$ is the fraction of nodes in G whose degree is equal to k.

Definition 4 (*Complementary cumulative degree distribution*) The complementary cumulative degree distribution of network G, $CCD(k)$, is the probability of observing a node with degree greater than or equal to k.

In directed networks each node has *in-degree* (the number of links pointing to a node) and *out-degree* (the number of links emanating from a node). The degree of the node is then the sum of its in-degree and out-degree, and is often called total-degree in order to emphasize the directed nature of the network. Consequently, for directed networks we have three degree distributions describing the connectivity of nodes: in-, out- and total-degree distributions.

Since class collaboration networks do not contain parallel links we can state the following:

1. The in-degree of the node representing class A is the number of other classes that use A, i.e. it is a measure of afferent coupling of A. If the in-degree of A is higher than 1 then A is being internally reused (used by more than one class). In other words, the in-degree of A reflects the degree of internal reuse of A.
2. The out-degree of class A is the number of other classes used (aggregated) by A, i.e. it is a measure of efferent coupling of A.

3. The total-degree of class A is the number of other classes to which A is coupled either through afferent or efferent coupling. This measure is equivalent to the well-known Coupling Between Objects (CBO) software metric from the Chidamber–Kemerer OO metric suite [7].

Many real-world networks belong to a class of heterogeneous networks characterized by a heavy-tailed, often power-law degree distribution [1,4,21], which means that the distribution has a long right tail of values that are far above the average degree.

Definition 5 (*Power-law*) Discrete probability distributions of the form $P(k) = Ck^{-\gamma}$, where C and γ are constants, are said to follow a power-law. The constant γ is called the scaling exponent of the power-law. Smaller γ implies slower decay of $P(k)$ and consequently causes a more skewed distribution. The constant C is determined by the normalization requirement that $\sum_{k=k_{min}}^{\infty} P(k) = 1$.

Definition 6 (*Scale-free network*) Networks whose (total) degree distributions follow a power-law in the tail, $P(k) \sim Ck^{-\gamma}$, are known as scale-free networks.

3 Related work

Valverde et al. [27] reported the first empirical evidence of scale-free and small-world properties in software systems. The authors examined the degree distributions, the small-world and clustering coefficients of undirected projections of class collaboration networks associated with Java Development Kit (JDK) version 1.2 and UbiSoft ProRally (computer game).

Myers [20] examined class collaboration networks representing 3 software systems written in C++ and static call graphs representing 3 software systems written in C. He computed the complementary cumulative in-degree and out-degree distributions reporting that these distributions have a power-law scaling region.

De Moura et al. [10] investigated properties of networks associated with four open source software projects written in C/C++. Similarly to previous studies, the authors reported that analyzed networks exhibit scale-free and small-world properties.

Hylland-Wood et al. [14] analyzed software networks of two Java open source software systems for a 15-month period of development. The authors constructed monthly snapshots of the networks at the package, class and method level and investigated properties of their in-degree and out-degree distributions. The results showed that the distributions follow truncated power-laws for each evolutionary snapshot. The study by Jenkins et al. [15] which examined properties of four Java class collaboration networks also indicated that the scale-free property in software systems is persistent across subsequent software releases.

Louridas et al. [17] analyzed a dataset of nineteen software networks that includes Java class collaboration networks. They found that in- and out-degree distributions of examined networks can be approximated by power laws, concluding that the scale-free property in software systems appears at various levels of abstraction, in diverse systems and languages.

Wheeldon and Counsell [29] examined statistical properties of software networks that represent different forms of OO coupling. The networks used in the study were extracted from three Java software systems. The results of the analysis showed that power-law scaling behavior characterizes different forms of class coupling. Baxter et al. [3] extended the study of Wheeldon and Counsell to a larger corpus of software networks associated with 56 Java software systems. In contrast to all previously mentioned studies, the authors considered power-law, log-normal and stretched exponential distributions to model empirically observed degree distributions.

Concas et al. [9] presented a comprehensive statistical analysis of an implementation of the Smalltalk system. Computed complementary cumulative distributions were tested against power-law and log-normal distributions. The parameters of the theoretical distributions were determined using maximum likelihood estimators, while the Pearson's χ^2 test was used to assess the quality of the fits. The authors found that the in-degree distributions show a power-law behavior in the tails, while the out-degree distributions exhibit log-normal behavior.

Taube-Schock et al. [26] examined 97 networks associated with software systems written in Java. The unique characteristic of their work is that the networks were constructed to encompass not only architectural elements as nodes, but also statements. The authors examined degree distributions of networks concluding that all of them are heavy-tailed. However, the decision to include statements in the analyzed networks can be considered as problematic. Statements can not be referenced and typically reference a low number of methods through method calls that are part of the statement. Since the number of statements in any large-scale software is drastically higher than the number architectural elements, the resulting network will have a vast majority of nodes with a low degree. Therefore, a heavy-tailed degree distribution of the network will be practically caused by the existence of nodes representing statements, not by the structure of dependencies among architectural elements.

4 Methodology to analyze high structural class coupling

Our methodology to analyze high structural class coupling is based on the notion of *enriched class collaboration networks*. Enriched class collaboration networks are class collaboration networks whose nodes are enriched with metric vectors containing both software metrics, as well as domain-independent metrics used in analysis of complex networks. In one of our previous works we presented SNEIPL [24]—a language-independent extractor of networks representing dependencies between source code entities. SNEIPL relies on a language-independent, enriched Concrete Syntax Tree (eCST) representation of source code [22]. From an input set of eCSTs produced by ANTLR-generated source code parsers, SNEIPL forms a variety of software networks, also including class collaboration networks and their subsets restricted to a particular coupling type (e.g. class inheritance). To support the methodology proposed in this paper, we extended the SNEIPL tool to form enriched class collaboration networks. More specifically, SNEIPL was extended to compute:

– domain-independent node centrality metrics (betweenness centrality and page rank) and object-oriented design metrics from the Chidamber–Kemerer metric suite [7]. Those metrics are calculated from class collaboration and class inher-

itance networks which the tool extracts from the eCST representation of source code.
- software metrics of class complexity (lines of code, cyclomatic complexity, etc.) that are calculated directly from the eCST representation of source code.

Our methodology to analyze high structural class coupling consists of four general steps:

1. The identification of weakly connected components in an enriched class collaboration network in order to isolate its giant weakly connected component (GWCC).
2. The degree distribution analysis of the GWCC relying on descriptive statistics and the power-law test introduced by Clauset et al. [8].
3. The analysis of highly coupled nodes in the GWCC relying on the metric-based comparison test in order to determine distinctive characteristics of hub classes.
4. The analysis of in-degree and out-degree of highly coupled nodes in the GWCC in order to determine weather high coupling was caused dominantly by internal class reuse or by internal class aggregation.

4.1 Connected component analysis

Weakly connected components in directed networks can be determined using fundamental graph traversal algorithms: depth-first search or breadth-first search. The existence of a GWCC in the analyzed class collaboration network implies that we are analyzing a software system that does not contain a large number of unused classes and/or small, functionally isolated groups of loosely coupled classes. In other words, small size weakly connected components in the class collaboration network correspond to software components composed of loosely coupled classes that are either (1) components in an early phase of development and not yet functionally integrated into the system, or (2) deprecated components which are not removed from the source code distribution. In both cases they can be omitted in further analysis since they are not coupled to highly coupled classes which, if exist, are located in the GWCC.

On the other hand, the absence of a GWCC indicates one of the following two scenarios:

- The whole system is an early phase of development that is conducted in a bottom-up manner, i.e. the software system is growing from independent components whose integration into fully functional system is not yet completed.
- The software system provides a set of unrelated functionalities.

In both of the above mentioned cases, the subsequent steps of our methodology should be performed separately for each of the weakly connected components that are large enough to ensure statistically valid findings.

4.2 Analysis of degree distributions

The connectivity of nodes in a network can be summarized by the degree distribution—the probability distribution of node degrees over the whole network. Knowing statistical properties of degree distributions of class collaboration networks is useful from both the theoretical and practical perspective. More precisely, statistical properties of degree distributions of class collaboration networks can motivate and guide theoretical principles behind predictive models of class coupling evolution. On the other side, software engineering practitioners can rely on parameters of fitted theoretical probability distributions in order to assess the design quality of software systems with respect to the principle of low coupling. For example, if the degree distribution of a class collaboration network follows a power-law $P(k) \sim k^{-\gamma}$ (as frequently reported in the literature, see Sect. 3) then the scaling exponent γ can be effectively used as an indicator of software design quality—smaller γ implies a more skewed distribution of class coupling which means a higher deviation from the principle of low coupling.

Regarding the high class coupling phenomenon, the most important descriptive statistics of empirically observed degree distributions are:

- Mean degree (μ)—the average number of in-/out-/total links incident to a node.
- Coefficient of variation (c_v)—a normalized measure of dispersion defined as the ratio of the standard deviation and the mean degree.
- Skewness (G_1)—the third standardized moment of the distribution which quantifies its asymmetry. Skewness equal to 0 implies that the distribution is perfectly symmetric (e.g. the normal distribution). Negative skewness indicates that the left tail of the distribution is longer than the right tail, while positive skewness indicates the opposite.

The absence of highly coupled classes implies that extreme values of node degree are absent. In other words, all nodes in the network have in-/out-/total degree that is close to the mean and, consequently, the network can be modeled by the Erdős-Renyi (ER) model of random graphs [12]. The basic characteristic of large-scale, finite size ER random graphs is that their degree distributions can be well approximated by the Poisson distribution. The Poisson distribution has coefficient of variation and skewness that are equal to $1/\sqrt{\mu}$. Therefore, if the coefficient or variation and skewness of empirically observed degree distributions are drastically higher than $1/\sqrt{\mu}$ then we can conclude that the analyzed network drastically deviates from the ER model of random graphs, further implying that it contains highly coupled nodes.

In recent years, researchers investigated a variety of real-world software systems under the framework of complex network theory, revealing the presence of the scale-free property in software networks. However, a common point in the majority of these studies is that the empirically observed degree distributions were tested only against a power-law. In only a few studies ([3,28] and [9]) distributions different than the power-law were additionally considered as theoretical models. The emergence of power-laws in evolving complex networks is most commonly explained by the principle of linear preferential attachment [2]. In order to use the linear preferential attachment principle in the prediction of class coupling evolution we have to exclude other two complementary principles governing evolution of complex networks: non-linear preferential attachment (the attachment probability depends on current class coupling but not linearly) and uniform attachment (the attachment probability does not depend on current class coupling). The uniform attachment principle results in networks whose degree distributions are exponential [2]. Super-linear preferential attachment leads to star networks that are very unlikely to occur for real-world software systems. Finally, sub-linear preferential attachment results in networks whose degree distributions are log-normal [23]. Therefore, in our methodology we propose investigation of statistical properties of degree distributions by testing them against power-law, exponential and log-normal distributions.

To determine parameters of previously mentioned theoretical distributions and assess the quality of obtained fits we rely on the power-law test introduced by Clauset et al. [8] which is currently the best performing test to validate power-laws in empirical data that considers also other theoretical models different than power-law. The test consists of the following three steps:

1. The scaling parameter of a power-law (α) is determined using the maximum likelihood estimation (MLE) with respect to some lower bound of the power-law behavior in the data (denoted by x_m). x_m is determined by the minimization of the weighted Kolmogorov–Smirnov (KS) statistic.
2. A large number of power-law synthetic data is generated using the estimated values of x_m and α in order to compute the goodness of the power-law model. The quality of the power-law fit (p value) is the probability that a randomly selected synthetic dataset has a higher value of the weighted KS statistic compared to the value of the same statistic for real data.
3. The parameters of alternative distributions are also determined using the appropriate MLEs. The power-law fit is compared to the fits of alternative distributions using the likelihood ratio test.

4.3 Metric-based analysis of highly coupled classes

There is one obvious difference between highly and loosely coupled classes—their coupling. Knowing other differences between those two categories of classes may be very helpful for a deeper understanding of the high class coupling phenomenon. Knowing distinctive characteristics of highly coupled classes may also be very useful for software engineering practitioners. For example, if highly coupled classes tend to have more complex control-flow compared to loosely coupled classes, then software developers should check whether they went through proper white-box testing before internally reusing them. Another example is related to the position of highly coupled classes in the class hierarchy: if highly coupled classes tend to have lower depth in the inheritance tree, then their refactoring may cause ripple effects through the whole class hierarchy.

Characteristics of OO classes can be quantitatively expressed not only by OO software metrics, but also by domain-independent, network-based metrics. For example, the functional importance of a class can be estimated by global centrality metrics used in social network analysis. Our methodology relies on two such metrics, betweenness centrality [13,16] and page rank [6], which are suitable and unbiased centrality indices for directed networks.

Definition 7 (*Betweenness centrality, BC*) The betweenness centrality of node z in directed network G, denoted by $C_b(z)$, is the extent to which z is located on the shortest paths connecting two arbitrary nodes different than z:

$$C_b(z) = \sum_{x,y \in V, x \neq y \neq z} \frac{\sigma(x, y, z)}{\sigma(x, y)},$$

where V is the set of nodes of G, $\sigma(x, y)$ is the total number of shortest paths connecting x and y, and $\sigma(x, y, z)$ is the total number of shortest paths connecting x and y that pass through z.

Definition 8 (*Page rank, PR*) The page rank of node z in directed network G, denoted by $PR(z)$, is given by the following recurrence relation:

$$PR(z) = \frac{1-\alpha}{n} + \alpha \sum_{i=1}^{n} A_{iz} \frac{PR(i)}{k_{out}(i)},$$

where A is the adjacency matrix of G, α is a constant called damping factor (usually set to 0.85), and $k_{out}(i)$ represents the out-degree of node i.

To determine distinctive characteristics of highly coupled nodes in enriched class collaboration networks we propose the metric-based comparison test (see Algorithm 1). To apply the test, each node in the network has to be characterized by a metric vector that minimally contains previously mentioned domain-independent centrality metrics and the following software metrics:

1. Metrics of volume and internal complexity. Those measures reflect how "big" and complex classes are. Metrics belonging to this category are: LOC (lines of code), CC (the cyclomatic complexity metric proposed by McCabe [19]), NUMA (the number of attributes in a class) and NUMM (the number of methods in a class).
2. Metrics of class coupling: in-degree (IN, afferent class coupling), out-degree (OUT, efferent class coupling) and total-degree (CBO) of nodes in the class collaboration network.
3. Inheritance metrics from the Chidamber–Kemerer metric suite: NOC (the number of children—the number of classes extending a class measuring the degree of internal reuse of the class through inheritance), and DIT (the depth of a class in the inheritance tree, measuring the degree of specialization of the class).

The metric-based comparison test is based on the application of the Mann–Whitney U (MWU) test [18]. The MWU test is a non-parametric statistical procedure that can be used to check whether values (scores) in one group tend to be systematically greater (or smaller) than the values in another group, when the values of both groups are not normally distributed and groups are not of equal size. The metric-based comparison test uses a binary classifier (HC) which separates hub nodes from non-hub nodes. Then for each metric from the metric vector it forms two sets of metric values: G_1—metric values for hub nodes, and G_2—metric values for non-hub nodes. After that, the MWU test is applied to G_1 and G_2. The MWU test checks the null hypothesis that the values in G_1 do not tend to be systematically (stochastically) greater or smaller than the values in G_2. The effect size of the Mann–Whitney test can be quantified by the so called probability of superiority (PS) [11]. We recorded two probabilities of superiority that were computed in a straightforward manner (by comparing each value from G_1 to each value from G_2):

Algorithm 1: Metric-based comparison test

input : N (class collaboration network), HC (hub classifier)

for *each metric* M **do**
  $G_1 :=$ empty set
  $G_2 :=$ empty set
  for *each node* n *in* N **do**
    if n *is hub according to* HC **then**
      $G_1 := G_1 \cup \{M(n)\}$
    else
      $G_2 := G_2 \cup \{M(n)\}$
  $p :=$ apply the MWU test to G_1 and G_2
  $ps_1 := 0$
  $ps_2 := 0$
  for *each* g_1 *in* G_1 **do**
    for *each* g_2 *in* G_2 **do**
      if $g_1 > g_2$ **then**
        $ps_1 := ps_1 + 1$
      else if $g_2 > g_1$ **then**
        $ps_2 := ps_2 + 1$
  $PS_1 := ps_1/(|G_1| \cdot |G_2|)$
  $PS_2 := ps_2/(|G_1| \cdot |G_2|)$
  if $p \geq 0.05$ **then**
    No statistically significant difference between hub nodes and non-hub nodes regarding metric M
  else
    Statistically significant difference between hub nodes and non-hub nodes regarding metric M
    if $PS_1 \geq 0.75$ **then**
      Hubs tend to have systematically higher values of M
    else if $PS_2 \geq 0.75$ **then**
      Non-hubs tend to have systematically higher values of M

- $\mathrm{PS}_1 = P\{X > Y\}$—the probability that a randomly selected value from G_1 (denoted by X) is larger than a randomly selected value from G_2 (denoted by Y).
- $\mathrm{PS}_2 = P\{Y > X\}$—the probability that a randomly selected value from G_2 is larger than a randomly selected value from G_1.

If the null hypothesis of the MWU test is rejected for metric M and $PS_1 \geq 0.75$ then we can conclude that hub nodes tend to have systematically higher values of metric M compared to non-hub nodes. The correctness of the metric-based comparison test directly follows from the correctness of the MWU test. The time complexity of the test is $O(mn^2)$ where m is the dimension of metric vector and n is the number of nodes in the network.

To apply the metric based comparison test it is necessary to define a hub classifier HC which separates hub classes from non-hub classes. In other words, we need to give a precise, mathematical definition of what is meant by highly coupled class. In the context of our methodology, we propose an approach that takes into account the size of the analyzed system.

Definition 9 (*Highly coupled class, hub*) Let V be the set of nodes in a class collaboration network, and let H denote a *minimal* subset of V which satisfies the following condition:

$$\sum_{h \in H} \text{degree}(h) > \sum_{o \in V \setminus H} \text{degree}(o).$$

Then, a class is considered highly coupled or hub if it belongs to the H set.

In other words, a class is considered highly coupled if it belongs to the minimal set of classes whose total CBO is higher than the total CBO of the rest of classes contained in the system. It is not hard to see that the H set can be computed in a straightforward manner using an appropriate greedy algorithm.

4.4 Analysis of afferent and efferent coupling of hubs

In the previous section we described the metric-based comparison test which can be employed to determine distinctive characteristics of highly coupled classes compared to loosely coupled classes. However, highly coupled classes can also be significantly different among themselves. We can distinguish between three categories of highly coupled classes:

1. Highly coupled classes whose coupling was dominantly caused by extensive internal reuse. Those classes are in class collaboration networks represented by nodes having high in-degrees that are close to their total degrees (or, equivalently, CBO values).
2. Highly coupled classes whose coupling was dominantly caused by internal aggregation of other classes present in the system. Classes belonging to this category are represented by nodes having high out-degrees that are close to their total degrees.
3. Highly coupled classes whose coupling was caused by both extensive internal reuse and aggregation, i.e. classes having both high in-degree and out-degree in class collaboration networks.

If the coupling of highly coupled classes tends to be dominantly caused by internal reuse, then the presence of high coupling in the system can indicate only the negative aspect of extensive internal reuse, not the negative aspects of extensive internal aggregation. On the opposite side, high coupling dominantly caused by extensive internal aggregation can indicate only the negative aspects of extensive internal aggregation, not the negative aspects of extensive internal reuse. To detect whether high coupling in a software system was dominantly caused by internal class reuse or internal class aggregation we propose the following metric named afferent–efferent coupling balance.

Definition 10 (*Afferent–Efferent Coupling Balance*) The afferent–efferent coupling balance, denoted by C_k, is the average ratio of in-degree to total degree (CBO) for classes whose total degree is higher or equal to k, i.e.

$$C_k = \frac{\sum_{i \in H_k} k_{in}(i)/k(i)}{|H_k|} = 1 - \frac{\sum_{i \in H_k} k_{out}(i)/k(i)}{|H_k|},$$

where H_k denotes the subset of nodes in the network whose total degree is higher than or equal to k, while $k_{in}(i)$, $k_{out}(i)$ and $k(i)$ are in-, out- and total degree of node i, respectively.

C_k is a normalized measure taking values in the range [0, 1] since in-degree is always smaller than or equal to total-degree. If $C_k = 1$ for some k then all classes having $CBO \geq k$ have out-degree that is equal to zero, which means that their coupling is caused entirely by internal class reuse. On the opposite side $C_k = 0$ implies that the coupling of all classes having $CBO \geq k$ is caused entirely by internal aggregation of other classes.

Definition 11 (*Afferent–Efferent Coupling Balance Plot*) An afferent-efferent coupling balance plot is the graph showing C_k for $k \in [m, M]$ where m and M are the minimal and maximal total degree of highly coupled classes (see Definition 9).

The afferent–efferent coupling balance plot shows the change of C_k for highly coupled classes. If C_k tends to have a high value which increases with k then we can conclude that internal class reuse is the prevalent cause of high structural class coupling. This further implies that software maintainers should take appropriate actions to prevent or reduce the negative aspects of extensive internal class reuse if highly coupled classes tend to be a source of problems during software evolution. On the opposite side, software maintainers should be aware of negative aspects of extensive class aggregation when C_k tends to have a low value which decreases with k. In the worst case, when C_k tends to take mid-range values and changes independently from k, software maintainers should act in a way to prevent the negative effects of both extensive class reuse and extensive class aggregation.

5 Experiments and results

Using the methodology described in the previous section we analyzed the high coupling phenomenon in five open-source, large-scale software systems written in Java (each of them having more than 10^6 LOC in total). The analyzed software systems are:

1. Apache Tomcat, a servlet container that implements the official Java Servlet and JavaServer Pages specifications,
2. Apache Lucene, a search engine library written in Java,
3. Apache Ant, a Java-based build tool,
4. Apache Xerces, a Java XML parsing library,
5. JFreeChart, a Java framework for creating and displaying charts.

Table 1 Experimental dataset of class collaboration networks

Software system	Version	LOC	N	L
Tomcat	7.0.29	329,924	1494	6841
Lucene	3.6.0	111,763	789	3544
Ant	1.9.2	219,094	1175	5521
Xerces	2.11.0	216,902	876	4775
JFreeChart	1.0.17	226,623	624	3218

N is the number of nodes, while L is the number of links

Table 2 The basic characteristics of empirically observed degree distributions

Software system	Distribution	μ	σ	c_v	G_1	M
Tomcat	Total-degree	9.58	15.93	1.66	8.05	293
	In-degree	4.79	13.86	2.90	11.49	293
	Out-degree	4.79	7.07	1.48	3.36	73
Lucene	Total-degree	9.02	11.58	1.28	5.95	175
	In-degree	4.51	9.84	2.18	7.22	153
	Out-degree	4.51	5.30	1.17	2.93	46
Ant	Total-degree	9.46	24.73	2.62	15.25	534
	In-degree	4.73	23.57	4.99	16.53	533
	Out-degree	4.73	5.46	1.16	2.38	40
Xerces	Total-degree	10.92	14.23	1.30	2.87	106
	In-degree	5.46	10.97	2.01	4.22	105
	Out-degree	5.46	8.71	1.60	3.84	91
JFreeChart	Total-degree	10.54	16.02	1.52	5.78	211
	In-degree	5.27	14.01	2.66	7.95	211
	Out-degree	5.27	7.48	1.42	4.39	93

μ—mean, σ—standard deviation, c_v—the coefficient of variation, G_1—skewness, M—maximal value

The basic structural characteristics of enriched class collaboration networks corresponding to previously mentioned software systems are summarized in Table 1. The networks were extracted using the SNEIPL tool [24]. It is important to emphasize that the selected software systems are exactly those that are used to validate the correctness and completeness of SNEIPL. In other words, the experimental dataset consists of networks which were shown to be highly similar to those extracted by another tool (DependencyFinder) and from a different data source (Java bytecode) [24].

The size of the largest weakly connected component for networks from our dataset varies from 92 to 99% of the number of nodes implying that each of them has a giant connected component. In each network the fraction of isolated nodes is extremely small (less than 3%) indicating that in examined software systems the presence of unused classes is reduced to a minimum. For each giant weakly connected component of the class collaboration networks from our experimental dataset we computed corresponding in-degree, out-degree and total-degree distributions. Table 2 shows the basic quantities describing empirically observed degree distributions.

Table 3 The results of the power-law test

System	Distr.	x_m	L	α	p value	R_{ln}	$p(R_{ln})$	R_e	$p(R_e)$
Tomcat	Total	17	276	2.8	0.99	−0.68	0.49	1.95	0.05
	In	4	289	2.07	0.74	−1.84	0.06	3.32	0.0009
	Out	19	54	3.69	0.46	−1.07	0.28	−0.75	0.45
Lucene	Total	10	165	2.68	0.64	−1.39	0.16	1.24	0.21
	In	3	150	1.95	0.17	−2.76	0.005	1.75	0.08
	Out	11	35	3.72	0.84	−0.29	0.76	1.28	0.19
Ant	Total	11	523	2.6	1.00	0.41	0.68	2.77	0.005
	In	7	526	2.08	0.78	−0.38	0.7	3.36	0.0007
	Out	14	26	4.17	0.42	−0.94	0.34	−0.29	0.76
Xerces	Total	49	57	4.65	0.83	−0.75	0.45	−0.85	0.39
	In	6	99	2.1	0.60	−2.45	0.01	1.18	0.23
	Out	27	64	4.29	0.66	−0.22	0.81	0.53	0.59
JFreeChart	Total	14	197	2.65	0.88	−0.47	0.63	2.38	0.01
	In	9	202	2.31	0.94	−0.4	0.68	2.31	0.02
	Out	14	79	3.79	0.14	−0.07	0.94	1.16	0.24

The obtained values of the coefficient of variation, skewness and maximal degree indicate the presence of hubs in the networks—classes whose in/out/total degree is significantly higher than the average degree. The coefficient of variation and skewness of degree distributions of Erdős-Renyi random graphs are equal to $1/\sqrt{\mu}$ where μ is the mean degree. Therefore, the coefficient of variation and skewness of degree distributions of Erdős-Renyi random graphs comparable to our networks are smaller than 0.5 ($\mu > 4$ for each empirically observed degree distribution). The coefficient of variation of empirically observed degree distributions is always higher than 0.5, while the skewness is drastically higher than 0.5 implying the presence of strong hubness in the analyzed class collaboration networks.

Using the power-law test proposed by Clauset et al. [8] (whose implementation is provided by the poweRlaw R package[1]) we investigated whether the degree distributions of examined software systems follow power-laws and consequently exhibit the scale-free property. Additionally, the test compares the best power-law fit to the best fits of the exponential probability mass function (PMF) and log-normal PMF in the obtained power-law scaling region. The results of the test are summarized in Table 3. As can be observed, power-laws are plausible models for all examined degree distributions since obtained p values are always higher than 0.1.

The results of the likelihood ratio tests which compare the power-law fits to the best fits of the log-normal and exponential PMFs in obtained power-law scaling regions are also shown in Table 3. The value of the log likelihood ratio is denoted by R_d in Table 3, where d is an alternative distribution ("ln"—log-normal, "e"—exponential). Positive and statistically significant R_d ($R_d > 0$, $p(R_d) < 0.1$) indicates that the

[1] http://cran.r-project.org/web/packages/poweRlaw/index.html.

Table 4 The results of the power-law test through the whole range of values ($x_m = 1$)

System	Distr.	$R\left(\frac{\text{pw}}{\text{ln}}\right)$	p	$R\left(\frac{\text{pw}}{\text{e}}\right)$	p	$R\left(\frac{\text{ln}}{\text{e}}\right)$	p
Tomcat	Total	−19.56	$<10^{-4}$	−7.80	$<10^{-4}$	3.81	0.0001
	In	−5.04	$<10^{-4}$	5.77	$<10^{-4}$	6.64	$<10^{-4}$
	Ou	−12.88	$<10^{-4}$	−6.23	$<10^{-4}$	3.78	0.0002
Lucene	Total	−19.66	$<10^{-4}$	−14.44	$<10^{-4}$	3.86	0.0001
	In	−4.68	$<10^{-4}$	4.70	$<10^{-4}$	6.12	$<10^{-4}$
	Out	−11.88	$<10^{-4}$	−10.12	$<10^{-4}$	0.31	0.7578
Ant	Total	−18.89	$<10^{-4}$	−4.24	$<10^{-4}$	3.26	0.0011
	In	−3.06	0.0022	4.24	$<10^{-4}$	4.61	$<10^{-4}$
	Out	−13.63	$<10^{-4}$	−11.27	$<10^{-4}$	−0.88	0.3807
Xerces	Total	−18.48	$<10^{-4}$	−10.86	$<10^{-4}$	5.82	$<10^{-4}$
	In	−5.08	$<10^{-4}$	7.00	$<10^{-4}$	9.15	$<10^{-4}$
	Out	−10.19	$<10^{-4}$	−2.06	0.0397	5.37	$<10^{-4}$
JFreeChart	Total	−16.20	$<10^{-4}$	−9.28	$<10^{-4}$	4.07	$<10^{-4}$
	In	−4.51	$<10^{-4}$	3.57	0.0004	4.94	$<10^{-4}$
	Out	−8.95	$<10^{-4}$	−3.40	0.0007	2.49	0.0127

power-law fit is favored over the best fit of the alternative distribution d. On the other hand, negative and statistically significant R_d ($R_d < 0$, $p(R_d) < 0.1$) implies that the alternative distribution better fits the tail of the distribution. If R_d is not statistically significant ($p(R_d) \geq 0.1$) then the best fits of both theoretical distributions are equally far from the empirically observed tail where the power-law fit is plausible. It can be seen that for all distributions that have a small power-law scaling region all considered theoretical distributions are equally plausible models. Moreover, all considered theoretical distributions are equally plausible models for the tail of the total degree distribution of the Lucene class collaboration network. For the rest of the distributions it can be concluded that:

- The best power-law fit is always better than the exponential fit except for the tail of the Xerces in-degree distribution where the log-normal PMF is the most plausible model.
- The power-law fit is never preferred over the log-normal fit. To the contrary, log-normal PMF is the better model for the tail of the Tomcat in-degree distribution, the Lucene in-degree distribution, and the Xerces in-degree distribution.

Using the likelihood ratio test we also investigated which theoretical distribution provides the best fit considering the whole range of degree values ($x_m = 1$). The results are summarized in Table 4. The value of the log likelihood ratio is denoted by $R(d_1/d_2)$, where d_1 and d_2 are two theoretical distributions ("pw"—power-law, "ln"—log-normal, "e"—exponential). Positive and statistically significant R ($R(d_1/d_2) > 0$, $p < 0.1$) indicates that d_1 is preferred over d_2, while a negative and significant value of R indicates the opposite. In the case when the value of R is not statistically significant

($p \geq 0.1$) then both distributions are equally plausible. As can be seen, the log-normal distribution provides the best fit to all distributions except for the out-degree distribution of Lucene and Ant where log-normal and exponential distributions are equally plausible.

Having in mind the definition of scale-free networks we can conclude that examined class collaboration networks are not scale-free for two reasons:

– The log-normal distribution provides a better fit to empirically observed degree distributions through the whole range of degree values, compared to the power-law.
– The tails of the distribution can be modeled by power-laws, but alternative distributions are either equally plausible models or even provide better fits.

We applied the metric-based comparison test with the previously defined hub classifier in order to detect metric differences between hubs and loosely coupled classes. The results are summarized in Table 5. For four software systems (all except Lucene) the null hypothesis of the Mann–Whitney U test is accepted only for the DIT (depth in inheritance tree) metric. This means that highly coupled classes tend to exhibit the same degree of specialization as loosely coupled classes. In all other aspects (voluminosity, internal complexity, degree of reuse and aggregation, centrality and importance) the differences between hubs and non-hubs are statistically significant. Drastic differences in all cases are present for the following metrics: LOC, BET and IN. This means that highly coupled classes tend to be drastically more voluminous, functionally important and internally reused compared to lowly coupled classes.

For Tomcat, Ant and JFreeChart, the drastic differences between hubs and ordinary classes can also be observed with respect to the NUMM (the number of methods) metric. This means that hubs in those systems tend to define drastically more methods than ordinary classes. In the case of Ant, we can observe a large difference between the cyclomatic complexity of hubs and the cyclomatic complexity of ordinary classes. This implies that highly coupled classes in Ant have significantly more complex control-flow implying that they are more difficult to test compared to non-hub classes.

Looking back to the data presented in Table 2 we can observe that the coefficients of variation, skewness and maximal degree of in-degree distributions are (drastically) higher than the same quantities describing out-degree distributions. This means that the tails of the in-degree distributions are longer ("heavier") than the tails of the out-degree distributions. The previous observation suggests that highly coupled classes, classes contained in tails of the total-degree distributions, tend to have higher in-degree than out-degree. Figure 1 shows the afferent–efferent coupling balance plots (see Definition 11) for the examined software systems. Additionally, each of the graphs shows the change of P_k, the probability that a randomly selected class whose total degree is higher than or equal to k has more two times higher in-degree than out-degree, with k. As can be observed for all software systems except Xerces, both C_k and P_k tend to increase with k. This means that the disbalance between the afferent and efferent coupling becomes more drastic with higher values of the CBO metric.

In the case of Xerces, C_k starts to decrease from 0.71 at $k = 50$ to 0.43 at $k = 96$. Highly coupled classes (CBO $>$ 50) in Xerces are dominantly caused by internal reuse ($C_{50} = 0.71$ and $P_{50} = 0.67$) but the magnitude of in-degree dominance decreased for higher values of CBO. This means that Xerces contains a significant

Table 5 The results of the metric-based comparison test

System	Metric	$\overline{C_1}$	$\overline{C_2}$	U	p	NH	PS_1	PS_2
Tomcat	LOC	699.03	137.29	204,480	$<10^{-4}$		**0.84**	0.15
	CC	71.47	13.69	176,951	$<10^{-4}$		0.68	0.22
	NUMA	12.18	3.89	178,509	$<10^{-4}$		0.7	0.22
	NUMM	31.68	7.21	206,308	$<10^{-4}$		**0.84**	0.13
	IN	20.36	2.35	198,789	$<10^{-4}$		**0.78**	0.14
	OUT	15.42	3.04	199,959	$<10^{-4}$		**0.8**	0.15
	NOC	1.25	0.18	136,294	0.006		0.18	0.05
	DIT	0.63	0.47	126,201	0.35	**acc**	0.27	0.23
	BET	4777.76	169.75	208,000	$<10^{-4}$		**0.83**	0.11
	PR	0.002055	0.000474	179,470	$<10^{-4}$		0.74	0.26
Lucene	LOC	332.5	93.03	69,044	$<10^{-4}$		**0.78**	0.22
	CC	37.11	13.32	62,822	$<10^{-4}$		0.66	0.25
	NUMA	7.52	2.93	59,358	$<10^{-4}$		0.62	0.28
	NUMM	15.17	5.91	65,950	$<10^{-4}$		0.72	0.23
	IN	16.26	2.04	76,210	$<10^{-4}$		**0.82**	0.11
	OUT	10.05	3.33	64,601	$<10^{-4}$		0.69	0.24
	NOC	1.68	0.26	55,865	$<10^{-4}$		0.35	0.09
	DIT	0.46	0.88	53,421	0.0002		0.2	0.4
	BET	2828.94	139.12	71,292	$<10^{-4}$		**0.76**	0.15
	PR	0.00347	0.000809	70,588	$<10^{-4}$		**0.79**	0.2
Ant	LOC	574.78	114.32	132,568	$<10^{-4}$		**0.86**	0.14
	CC	49.67	9.63	124,300	$<10^{-4}$		**0.78**	0.17
	NUMA	12.37	3.66	117,880	$<10^{-4}$		0.73	0.2
	NUMM	26.08	6.25	133,043	$<10^{-4}$		**0.85**	0.13
	IN	24.36	1.79	126,563	$<10^{-4}$		**0.78**	0.14
	OUT	13.30	3.42	129,639	$<10^{-4}$		**0.82**	0.14
	NOC	2.88	0.2	95,863	$<10^{-4}$		0.31	0.07
	DIT	1.28	1.11	83,411	0.11	**acc**	0.39	0.32
	BET	7715.53	249.41	129,761	$<10^{-4}$		**0.8**	0.12
	PR	0.003835	0.000411	117,779	$<10^{-4}$		**0.75**	0.23
Xerces	LOC	766.36	145.84	71,757	$<10^{-4}$		**0.79**	0.21
	CC	100.54	15.78	64,222	$<10^{-4}$		0.64	0.22
	NUMA	22.35	3.18	66,545	$<10^{-4}$		0.69	0.22
	NUMM	24.35	8.26	67,912	$<10^{-4}$		0.73	0.23
	IN	22.72	2.71	77,836	$<10^{-4}$		**0.83**	0.11
	OUT	17.67	3.51	66,649	$<10^{-4}$		0.7	0.23
	NOC	1.46	0.23	53,140	0.003		0.26	0.09
	DIT	0.68	1.07	50,199	0.06	**acc**	0.23	0.33

Table 5 continued

System	Metric	$\overline{C_1}$	$\overline{C_2}$	U	p	NH	PS_1	PS_2
	BET	3897.61	182.87	72,166	$<10^{-4}$		**0.76**	0.16
	PR	0.003367	0.000788	67,360	$<10^{-4}$		0.74	0.25
JFreeChart	LOC	854.12	202.85	37,753	$<10^{-4}$		**0.77**	0.23
	CC	59.83	12.32	36,467	$<10^{-4}$		0.71	0.22
	NUMA	11.97	3.35	33,972	$<10^{-4}$		0.65	0.27
	NUMM	37.25	10.02	38,177	$<10^{-4}$		**0.77**	0.21
	IN	21.45	2.34	39,365	$<10^{-4}$		**0.76**	0.16
	OUT	14.38	3.56	37,118	$<10^{-4}$		0.72	0.21
	NOC	1.77	0.23	31,594	$<10^{-4}$		0.35	0.06
	DIT	1	0.83	27,197	0.08	**acc**	0.39	0.27
	BET	931.58	67.69	39,678	$<10^{-4}$		**0.77**	0.14
	PR	0.004705	0.001066	35,461	$<10^{-4}$		0.71	0.26

$\overline{C_1}$—the average value of the corresponding metric for hubs, $\overline{C_2}$—the average value of the corresponding metric for non-hubs, U—the Mann–Whitney U statistic, p—p value of U, NH—if **acc** then the null hypothesis of the MWU test is accepted, PS_1—the probability of superiority of hubs over non-hubs (bold values indicate that hubs tend to have systematically higher values of the corresponding metric), PS_2—the probability of superiority of non-hubs over hubs

portion of extremely highly coupled classes that are either dominantly caused by internal aggregation or both aggregation and reuse significantly contribute to total coupling. The top ten most coupled classes in Xerces are shown in Table 6. It can be observed that the list contains:

- Two classes (XSDHandler and XMLSchemaValidator) whose coupling is dominantly caused by the internal aggregation of a large number of other classes.
- Four classes (XNIException, QName, SymbolTable and Constants) whose coupling is entirely or almost entirely caused by their excessive internal reuse.
- Four classes where both internal aggregation and internal reuse significantly contribute to total coupling.

6 Discussion

The analysis of five large-scale software systems using the methodology proposed in this paper firstly revealed that high structural class coupling in real software systems cannot be accurately modeled by power-law distributions. This observation has both theoretical and practical relevance. From the practical point of view, it implies that the scaling exponent of fitted power-laws can not be used by software engineering practitioners as a metric of software design quality. From the theoretical stand point, it means that generating mechanisms of power-law distributions cannot be used to guide and formulate theoretical models of class coupling evolution.

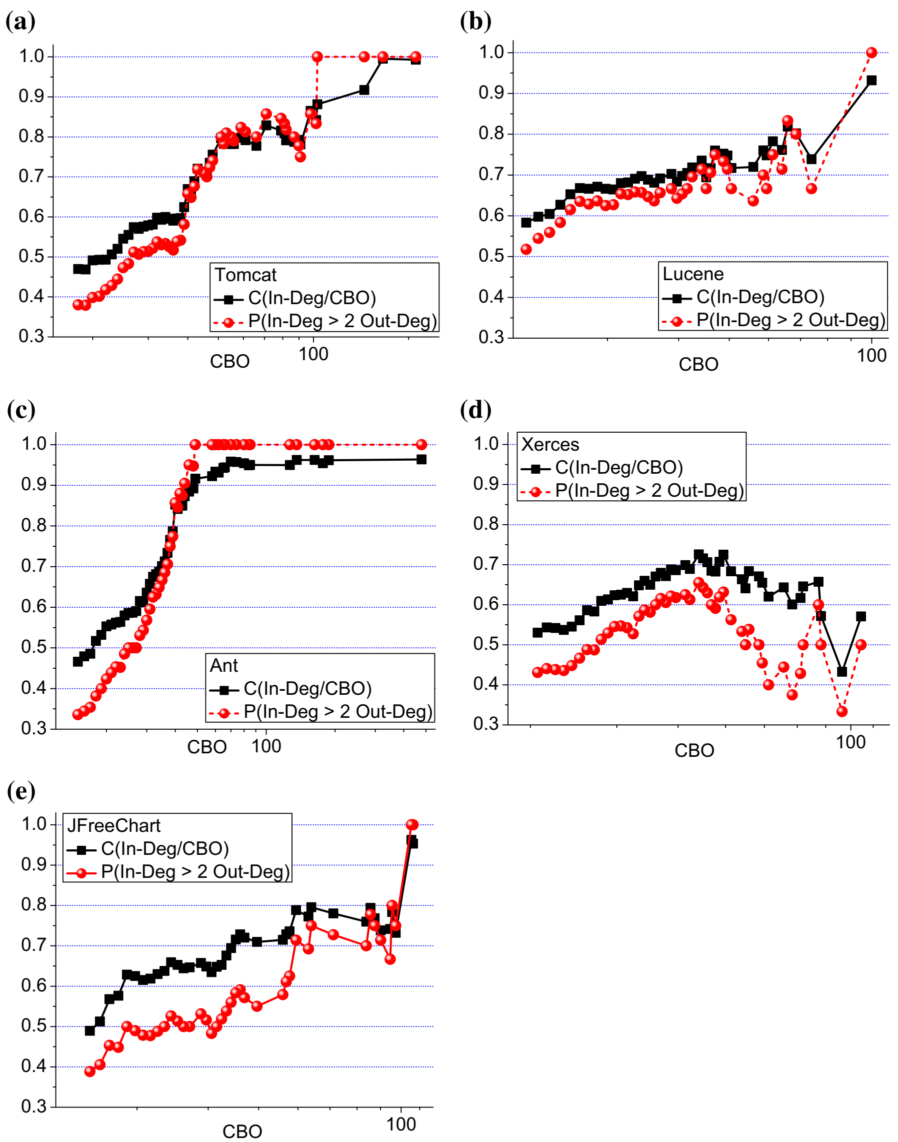

Fig. 1 The afferent–efferent coupling balance plot for **a** Tomcat, **b** Lucene, **c** Ant, **d** Xerces, and **e** JFreeChart

Degree distribution analysis also showed that empirically observed in-, out- and total-degree distributions are heavy-tailed distributions that can be modeled with the log-normal distribution. The near-linear preferential attachment principle, a generating mechanism for log-normal degree distributions, suggests the tendency of increasing internal reuse for already highly reused classes as the class collaboration network grows through software evolution. Software engineering practice encourages internal

Table 6 The top ten most coupled classes in Xerces

Class	In	Out	CBO
org.apache.xerces.impl.xs.traversers.XSDHandler	15	91	106
org.apache.xerces.xni.XNIException	105	0	105
org.apache.xerces.impl.xs.XMLSchemaValidator	15	81	96
org.apache.xerces.util.SymbolTable	86	1	87
org.apache.xerces.xni.QName	86	0	86
org.apache.xerces.dom.CoreDocumentImpl	47	33	80
org.apache.xerces.impl.xs.SchemaGrammar	35	44	79
org.apache.wml.dom.WMLDocumentImpl	37	39	76
org.apache.xerces.impl.Constants	72	1	73
org.apache.xerces.impl.XMLEntityManager	28	40	68

code reuse and this tendency may seem very desirable. However, modifications of a highly reused class may affect a very large number of classes which directly depend on it. In addition, highly reused classes are particularly critical (externally responsible) because any defect in classes with high degrees of internal reuse are more likely to propagate to other parts of the system [5]. In the case when defects need to propagate to other parts of the system and cause failures there in order to be detected, they may not be detected when testing the entity in isolation. Thus, identifying highly reused classes, especially ones which do not realize simple functionalities, and their effective testing or validation before they become extremely reused may help to prevent the following conflict situation: the presence of highly reused, hard-to-modify and hard-to-test classes with an increasing tendency of internal reuse which makes them even more critical and harder to maintain in terms of modifiability and use-context-required testability.

Similarly as for in-degree distributions, a heavy-tailed out-degree distribution implies a broad spectrum of internal aggregation among classes. As stated by Briand et al. [5], a high degree of internal aggregation of other classes by a class can cause its

- lower understandability due to high class aggregation—in order to fully comprehend a class that aggregates a large number of other classes one has also to examine and understand all aggregated classes,
- higher proneness to errors—the probability that some aggregated class contains a fault or that it is incorrectly used increases with the number of aggregated classes,
- lower external reusability—if a class aggregates a large number of other classes then its reuse in some other software project would require reuse of aggregated classes as well.

In software development practice, it is desirable to keep class coupling as low as possible. Heavy-tailed total degree distributions imply that coupling among classes has no characteristic scale: average class coupling is relatively small, but there is a statistically significant number of highly coupled classes whose degree of coupling is extremely large. From the software engineering perspective, this phenomenon is considered to be bad, because highly coupled entities can cause difficulties in software maintenance and program comprehension. The analysis of the balance between in-degree and out-degree for highly coupled classes in four Java software systems (all except Xerces) showed that the origin of their high coupling, which is as already mentioned considered as an indicator of poor software design, is in extensive internal reuse, which is, to the contrary, considered desirable in software development practice. This seems to be a paradox. However, high coupling caused by extensive internal reuse can indicate only negative aspects of extensive internal reuse, not negative aspects of extensive internal aggregation. In systems where hubs are caused predominantly by internal reuse, the existence of high coupling can suggest high criticality and lower maintainability in terms of context-required testability, but not lower maintainability in terms of lower understandability and higher error-proneness due to high class aggregation. In the case when highly reused classes tend to be simple (and thus problem-free) or when they are extensively tested (or validated) in early phases of software development and do not tend to change drastically during the evolution, we can actually consider high coupling caused by extensive internal reuse as an indicator of good rather than poor modularization. In such situations, high coupling means low redundancy of code and proper abstraction of highly reused classes. However, in the case when a highly reused class tends to be unstable during software evolution, in the sense that its modification forces modifications in a large number of classes that depend on it, then software maintainers need to control its coupling/internal reuse and keep it as low as possible. In the case of Xerces, high coupled classes are caused significantly by both internal class reuse and aggregation. Therefore, the maintainers of this system should be aware of the negative aspects of both extensive class reuse and extensive class aggregation.

The application of the metric-based comparison test showed that highly coupled classes in all examined systems tend to be drastically more voluminous (contain more lines of code) and more functionally important compared to loosely coupled classes. This observation directly implies that (1) the internal logic of highly coupled classes is harder to comprehend compared to loosely coupled classes and (2) changes in highly coupled classes may have bigger impact to the overall system stability and evolution. Additionally, in the case of Ant we observed that highly coupled classes tend to have systematically higher cyclomatic complexity than loosely coupled classes. This implies that highly coupled Ant classes require a significantly larger number of test cases for white-box testing due to their complex control flow. Therefore, we can conclude that Ant hubs are harder to test compared to loosely coupled classes. Having in mind that hubs in Ant are dominantly caused by extensive class reuse we can infer the following:

- Ant developers that intend to reuse Ant hub classes should be aware of their complex control flow and check whether reused classes went through proper white-box testing in order to avoid the situation that faults in their classes were caused by errors propagated from hub classes.
- Ant maintainers should be aware of the fact that faults associated to hub classes are most probably not caused by propagated errors (errors in classes referenced

by hub classes) since hub classes do not tend to aggregate a large number of other classes, but by errors in their complex control-flow.
– Ant project managers should be aware of the fact that hub classes are the most important classes in the system and that failures in these classes can have a big impact to the stability of the whole system. Therefore, they should initiate proper testing procedures if it is observed that hub classes tend to cause faults in other classes.

7 Conclusions and future work

In this paper we proposed an innovative, statistically robust, network-based methodology to examine high structural class coupling in object-oriented software systems. The main idea of the methodology is to enrich nodes of class collaboration networks with both software and domain-independent metrics proposed under the framework of complex network theory in order to perform a more comprehensive analysis of highly coupled classes.

Following the proposed methodology, we analyzed high class coupling in five open-source software systems written in Java. The obtained results showed that high class coupling in real software systems cannot be accurately modeled by power-law distributions, further implying that the scaling-exponent of power-laws is not a reliable metric of software design quality. Secondly, we showed that our methodology enables the identification of key differences between highly and loosely coupled classes that enhance the understanding of the high class coupling phenomenon in concrete software systems. Finally, the experimental results indicate that extremely highly coupled classes in real software systems are caused predominantly by internal reuse, and consequently that high coupling would rather indicate only negative aspects of internal class reuse, not negative aspects of internal class aggregation.

In future work we plan to extend this study to a larger experimental dataset also encompassing class collaboration networks representing large-scale software systems written in programming languages different than Java. Using the same methodological framework we will also investigate the coupling structure of software systems at different levels of abstraction such as those represented by package and method collaboration networks. Finally, our future work will be devoted to evolutionary analysis of the high class coupling phenomenon.

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

References

1. Albert R, Barabási AL (2002) Statistical mechanics of complex networks. Rev Mod Phys 74(1):47–97. doi:10.1103/RevModPhys.74.47
2. Barabasi AL, Albert R (1999) Emergence of scaling in random networks. Science 286(5439):509–512. doi:10.1126/science.286.5439.509

3. Baxter G, Frean M, Noble J, Rickerby M, Smith H, Visser M, Melton H, Tempero E (2006) Understanding the shape of Java software. In: Proceedings of the 21st annual ACM SIGPLAN conference on object-oriented programming systems, languages, and applications, OOPSLA ’06, pp 397–412. ACM, New York, NY, USA . doi:10.1145/1167473.1167507
4. Boccaletti S, Latora V, Moreno Y, Chavez M, Hwang D (2006) Complex networks: structure and dynamics. Phys Rep 424(45):175–308. doi:10.1016/j.physrep.2005.10.009
5. Briand LC, Daly JW, Wüst JK (1999) A unified framework for coupling measurement in object-oriented systems. IEEE Trans Softw Eng 25(1):91–121
6. Brin S, Page L (1998) The anatomy of a large-scale hypertextual web search engine. Comput Netw ISDN Syst 30(1–7):107–117
7. Chidamber SR, Kemerer CF (1994) A metrics suite for object oriented design. IEEE Trans Softw Eng 20(6):476–493. doi:10.1109/32.295895
8. Clauset A, Shalizi C, Newman M (2009) Power-law distributions in empirical data. SIAM Rev 51(4):661–703. doi:10.1137/070710111
9. Concas G, Marchesi M, Pinna S, Serra N (2007) Power-laws in a large object-oriented software system. IEEE Trans Softw Eng 33(10):687–708
10. de Moura APS, Lai YC, Motter AE (2003) Signatures of small-world and scale-free properties in large computer programs. Phys Rev E 68(1):017,102. doi:10.1103/PhysRevE.68.017102
11. Erceg-Hurn DM, Mirosevich VM (2008) Modern robust statistical methods: an easy way to maximize the accuracy and power of your research. Am Psychol 63(7):591–601. doi:10.1037/0003-066X.63.7.591
12. Erdös P, Rényi A (1959) On random graphs, I. Publ Math Debr 6:290–297
13. Freeman LC (1977) A set of measures of centrality based on betweenness. Sociometry 40:35–41
14. Hylland-Wood D, Carrington D, Kaplan S (2006) Scale-free nature of Java software package, class and method collaboration graphs. Technical report, TR-MS1286, MIND Laboratory, University of Maryland, College Park, USA
15. Jenkins S, Kirk SR (2007) Software architecture graphs as complex networks: a novel partitioning scheme to measure stability and evolution. Inf Sci 177:2587–2601. doi:10.1016/j.ins.2007.01.021
16. Kósa B, Balassi M, Englert P, Kiss A (2015) Betweenness versus linerank. Comput Sci Inf Syst 12(1):33–48. doi:10.2298/CSIS141101092K
17. Louridas P, Spinellis D, Vlachos V (2008) Power laws in software. ACM Trans Softw Eng Methodol 18(1):2:1–2:26
18. Mann HB, Whitney DR (1947) On a test of whether one of two random variables is stochastically larger than the other. Ann Math Stat 18(1):50–60. doi:10.2307/2236101
19. McCabe TJ (1976) A complexity measure. IEEE Trans Softw Eng 2(4):308–320. doi:10.1109/TSE.1976.233837
20. Myers CR (2003) Software systems as complex networks: structure, function, and evolvability of software collaboration graphs. Phys Rev E 68(4):046,116. doi:10.1103/PhysRevE.68.046116
21. Newman MEJ (2003) The structure and function of complex networks. SIAM Rev 45:167–256. doi:10.1137/S003614450342480
22. Rakić G, Budimac Z (2011) Introducing enriched concrete syntax trees. In: Proceedings of the 14th international multiconference on information society (IS), collaboration, software and services in information society (CSS), pp 211–214
23. Redner S (2005) Citation statistics from 110 years of Physical Review. Phys Today 58(6):49–54. doi:10.1063/1.1996475
24. Savić M, Rakić G, Budimac Z, Ivanović M (2014) A language-independent approach to the extraction of dependencies between source code entities. Inf Softw Technol 56(10):1268–1288. doi:10.1016/j.infsof.2014.04.011
25. Šubelj L, Bajec M (2012) Software systems through complex networks science: review, analysis and applications. In: Proceedings of the first international workshop on software mining, SoftwareMining ’12, pp 9–16. ACM, New York, NY, USA. doi:10.1145/2384416.2384418
26. Taube-Schock C, Walker R, Witten I (2011) Can we avoid high coupling? In: Mezini M (ed) ECOOP 2011 object-oriented programming. Lecture notes in computer science, vol 6813. Springer, Berlin, pp 204–228. doi:10.1007/978-3-642-22655-7_10
27. Valverde S, Cancho RF, Solé RV (2002) Scale-free networks from optimal design. EPL 60(4):512–517. doi:10.1209/epl/i2002-00248-2

28. Wen H, DSouza RM, Saul ZM, Filkov V (2009) Evolution of Apache open source software. In: Ganguly N, Deutsch A, Mukherjee A (eds) Dynamics on and of complex networks, modeling and simulation in science, engineering and technology. Birkhuser, Boston, pp 199–215. doi:10.1007/978-0-8176-4751-3_12
29. Wheeldon R, Counsell S (2003) Power law distributions in class relationships. In: Proceedings of the third IEEE international workshop on source code analysis and manipulation, pp 45–54. doi:10.1109/SCAM.2003.1238030
30. Yourdon E, Constantine LL (1979) Structured design: fundamentals of a discipline of computer program and systems design, 1st edn. Prentice-Hall Inc., Upper Saddle River