Journal of Object Technology
Published by AITO — Association Internationale pour les Technologies Objets, © JOT 2011
Online at http://www.jot.fm.

Parameter-based refactoring and the relationship with fan-in/fan-out coupling

Alessandro Murgia[a] Roberto Tonelli[a] Giulio Concas[a]
Michele Marchesi[a] Steve Counsell[b]

a. Department of Electrical and Electronic Engineering - University of Cagliari, Italy

b. Department of Information Systems and Computing - Brunel University, London, United Kingdom

Abstract In this paper, we analyze the effect of particular refactorings on class coupling for different aggregate releases of four object-oriented Open Source (OS) Java software systems: Azureus, Jtopen, Jedit and Tomcat, as representative of general Java OS systems. Specifically, the "add parameter" to a method and "remove parameter" from a method refactorings, as defined according to Fowler, may influence class coupling changing fan-in and fan-out of classes they are applied to. We investigate, both qualitatively and quantitatively, using a statistical approach, the global effect of the application of such refactorings, providing best fitting statistical distributions able to describe the changes in fan-in and fan-out couplings. Results show a net tendency of developers to apply such refactorings to classes with relatively high fan-in and fan-out and a persistence of the same statistical distribution for fan-in and fan-out before and after refactoring. Finally, we provide a detailed analysis of the best fitting parameters and of their changes when refactoring occurs, which may be useful for estimating the effect of refactoring on coupling before it is applied. Since refactoring requires time and effort, these estimates may help in determining costs and benefits.

Keywords Refactoring, Coupling, Metric distribution, Fan-in, Fan-out

1 INTRODUCTION

According to Fowler [3] refactoring is defined as *a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing* *its observable behavior.* One area of research that the refactoring community has yet to tackle is the effect that certain refactorings have on crucial system features such as coupling, when applied repeatedly over time [4, 3]. Practitioners are interested in the effect of refactoring from a practical point of view (i.e., how refactoring influences effectively the coupling) and cumulatively (i.e., how refactoring performed over years of software development may change the coupling). The authors know of no study that has looked at the longitudinal effect of refactorings from a coupling perspective. In OO systems, refactoring may reduce maintenance effort rendering interfaces less coupled and thus easier to handle and use. In this context, refactorings such as *add parameter* and *remove parameter* play a key role since they heavily determine how objects send messages to each other.

Fowler describes how parameter-based refactorings change the software "qualitatively", overlooking how they change software "quantitatively" [3]. As an example, when we apply *remove parameter* to the signature of a method (i.e., when a parameter is no longer used by the method body), we can potentially reduce its coupling. However, it is rare to see only one type of refactoring applied to a class. A more likely scenario is that when *remove parameter* is performed, an *add parameter* refactoring is applied at the same time. Any effect of the first refactoring is thus limited by the application of the second.

We believe refactoring helps to improve software design, but without knowing what impact refactoring has during software evolution, it is difficult to balance costs and benefits of performing such tasks. We face this challenge by focusing our analysis on the relationship between parameter-based refactorings and coupling. We measure coupling through fan-in and fan-out — the earliest software metrics used for such measure [5]. Fan-out is defined as the number of other classes referenced by a class; fan-in is defined as the number of other classes that reference a class. Parameter-based refactorings have "direct" impact on class fan-out and a side effect on class fan-in. This work investigates the "direct" and "indirect" effects of these refactorings, adding a contribution in understanding possible hidden effects. We chose fan-in and fan-out metrics in preference to Chidamber and Kemerer's Coupling Between Objects (CBO) [6] metric because CBO measures coupling irrespective of its direction and provides a rough-grained detail of incoming and outgoing coupling, respectively.

We use the Complementary Cumulative Distribution Function (CCDF) to characterize the statistical distributions of fan-in and fan-out metrics in refactored and non-refactored classes. We compute the parameters of the best-fit functions of such metrics and use them to infer statistical properties of refactored classes. Our primary goal is to understand how (parameter based) refactoring changes fan-in and fan-out coupling (RQ2, RQ4). The achievement of such a task requires prior analysis of fan-in and fan-out distributions of refactored and non-refactored classes. To understand if and how Add Parameter (AP) and Remove Parameter (RP) refactorings affect the class coupling distribution, one must know in advance how the class distribution is in general. For this reason, we provide a discussion related to distribution characteristics taking into account refactored classes, classes that have not been refactored and both (RQ1,RQ3). Comparing these sets makes it easier to pursue our main goal and highlight developer habits in performing such refactorings.

Analysis was driven by the following 4 research questions:

- RQ1- Are there analytical distribution functions able to describe fan-in in refactored and non-refactored classes? Are there differences among refactored and non-refactored classes?

- RQ2- Based on the results of RQ1, does refactoring change fan-in coupling overall?
- RQ3- Are there analytical distribution functions able to describe fan-out in refactored and non-refactored class? Are there differences among refactored and non-refactored classes?
- RQ4- Based on the results of RQ3, does refactoring change fan-out coupling overall?

The four research questions were developed using the GQM approach of Basili et al., [2], with a *goal* of investigating the impact of parameter based refactoring on coupling as systems evolve. These questions are embedded in RQ1-RQ4 and the metrics are fan-in and fan-out. The *focus* is to evaluate how software metrics change when parameter-based refactoring is performed with respect to their evolution where such refactorings are not performed. The *viewpoints* are those of a researcher, of a development team leader and of a manager. The first is more aware of the practical impact of these refactorings on coupling of real software systems; likewise, the second can take leverage of this knowledge deciding on parameter based refactoring maintenance, according to the relevance of their benefits. The latter, knowing fan-in and fan-out distributions for all system classes, can help with better estimation of rework costs due to refactored classes. The *environment* of this paper regards four OO systems: Tomcat, Jedit, Jtopen, Azureus.

Results from our analysis showed that in refactored classes and non-refactored classes fan-in and fan-out had different distributions, since they possess different best fitting parameters. We reveal that fan-in and fan-out metrics are distributed according to a power-law and lognormal distributions, respectively. This is also true in refactored classes, to a high degree of accuracy, and this shows a consistent trend across all releases delivered, for all systems analyzed. Most interestingly and counter to what one might expect, the data pointed to a tendency for developers to prefer refactoring classes with a relatively high fan-in and fan-out. The study thus characterizes classes targeted for these refactorings and how their coupling changes.

The remainder of the paper is organized as follows. Section 2 reports relevant related work. Section 3 summarizes background concepts on the used statistical distributions. Section 4 describes how we collected the data. Section 5 presents the results and Section 6 their interpretation. Section 7 discusses threats to the validity of the study. Finally, in Section 8 we present the conclusions.

2 RELATED WORK

Refactoring, introduced by Opdyke in his PhD. thesis [14], was investigated by Fowler [3] who proposed 72 refactorings in his seminal text and from which we draw on herein. Refactoring identification can been addressed from two different prospectives. On the one hand, it can be studied where the refectoring needs to be performed. Antoniol et al. used time series to predict refactorings [26] and Bodhuin et al. used a GA-led tool for suggesting refactorings [28]. Zhao and Hayes [15] used metrics to predict and prioritise classes that most needed refactoring. On the other hand, it can be studied where refactorings of software modules have already been done. Demeyer et al. [11] used change metrics as a basis for finding refactorings; whereas Biegel et al. [27] adopted different similarity measures to detect refactorings.

Our study identifies where refactorings have already been applied. For this task, we used Ref-Finder [1] which, differently from the others studies, uses a template-based refactoring reconstruction approach. This tool, being open-source, is useful for replicating research studies. The parameter based refactorings embodied in the tool were used for our analysis. The goal of many studies is the identification of similarities among refactored software modules. Zhao and Hayes [15] suggested that manual refactoring selection by Java programmers overlooked the difficult, more complex refactorings in favour of easier, smaller refactorings. Advani et al., [16] showed the same feature to be true where refactoring data across multiple releases of open-source were mined. Research showed the majority of refactorings were simpler refactorings and not complex structural refactorings. Counsell et al., [19] also investigated the most commonly used and least commonly used refactorings and the inter-dependencies of those refactorings; refactorings used rarely were those with many inter-dependencies (i.e., some refactorings made extensive use of other 'sub-refactorings'). Commonly applied refactorings were those with fewer inter-dependencies.

This paper tackles this research area characterizing refactored classes by means of fan-in and fan-out metrics. Mubarak et al. [13] investigated fan-in and fan-out from an open-source evolutionary perspective, but did not explore refactoring *per se*. The study found a range of explanations for why the fan-in and fan-out differed between classes over time, most noticeably through the existence of *key* classes. Previous research [12] by the same authors showed that the fan-in and fan-out metrics tended to be relatively small for classes removed from a system. In other words, classes with either high fan-in and/or fan-out may be difficult to move around or be removed from a system. Our paper complements the previous finding, investigating the relationship between class fan-in/fan-out and its propensity to be refactored.

This paper is an extension of the research reported in [32], where a single project is analysed. In this work we generalise the results by adding 3 more projects to the dataset: Azureus, Jedit and Jtopen. In this way we reduce threats to external validity. This work also extends the research in [31][33]. In fact, in this paper we provide not only an analysis of metric desctibution but also compute the best fitting parameters in each case. This allows us to support our conclusions with more robust analytical results. We have evaluated the R^2 between metric distributions and typical distributions used to represent software metrics. This contributes to the refactoring research field which lacks, as far as we know, a deep and quantitative analysis in such matters.

3 BACKGROUND

3.1 Statistical distributions

Foreseeing future values of software metrics is a crucial step in scheduling maintenance activities of software products. In this context, the analysis of analytical statistical distributions behind the empirical data plays a key role, providing theoretical models for explaining experimental data. Moreover, studying fan-in and fan-out distributions might help in the identification of a generative model describing such distributions[1];

[1]The distribution analysis according to Mullen [34] suggests a model able to reproduce the lognormal distribution of software failures. Mullen was driven by two observations: (1) a lognormal distribution arises when the value of a variable is determined by the multiplication of many random factors, (2) previous work in literature points out that failure rates are determined by a multiplicative process.

this can provide estimates about the effort required during maintenance of classes [24]. To describe the statistical behavior of a distribution with respect to a metric, one can compute its empirical Probability Distribution Function (PDF). However, much more information is contained in the Cumulative Distribution Function (CDF) (Eq. 1) and in the Complementary Cumulative Distribution Function (CCDF) (Eq. 2). In fact, they preserve all the information contained in the original data, while the empirical PDF needs a binning of the independent variable for building a histogram, cumulating different points into a single bin and losing the information carried by each single point. Defined by $p(x)$ the PDF, by $P(x)$ the CDF, and by $G(x)$ the CCDF, we have [7]:

$$P(x) = p(X \le x) = \int_{-\infty}^{x} p(x')dx' \tag{1}$$

$$G(x) = p(X \ge x) = \int_{x}^{\infty} p(x')dx' \tag{2}$$

with

$$G(x) = 1 - P(x) \tag{3}$$

The CCDF represents the probability that the metric, X, is greater than or equal to a given value x. The CCDF is also preferable for a major improvement of the quality of fit in the tail of distribution, reducing the statistical fluctuations related to the reduced number of points into the bins in the tail. Further detailed discussion may be found in [7].

Many software metrics show a “fat-tail” distribution, namely that a small part of the population takes over a large proportion of the measured resource. In other words, even if there are many classes with a small value of the metric, there are also other classes with a non-negligible probability of metric values several orders of magnitude higher than the average. For this reason, statistics like average and standard deviation cannot properly characterize fat-distributions [7]. Such fat-tails can usually be well represented by a power-law relationship, which, when plotted in a log-log scale, appears as a straight line. It is often convenient to use log-log plots in order to exploit such fat tail behaviour and we adhere to this practice in this work.

3.1.1 Power Law

The power-law distribution is expressed by the formula:

$$p(x) \simeq x^{-\alpha} \tag{4}$$

where α is a scaling parameter, or simply the power-law coefficient. This distribution is simply characterized using only the parameter α, besides a normalization factor. When plotted in a log-log plot, the power-law function is easy to spot because of the perfect straight line (with slope α). This statistical distribution is ubiquitous and is found in many different fields [29] [18].

Being the function divergent in the origin for $\alpha \ge 1$, it is not possible to report real data in its entire range of values. In addition, real data generally has a lower cut-off, indicated as x_0, above which the power-law holds. This value represents a second parameter used to achieve better fittings and may be useful to indicate if the data are power-law distributed from the beginning, or if a fat tail exists at the end of the distribution (which may be different from a true power-law in the remaining range). This situation may occur for lognormally distributed data. The algorithm we use for best fitting the empirical data with a power-law distribution estimates the lower cut-off x_0 and the power-law exponent α, according to the goodness-of-fit based method described in [8] and also provides the Kolmogorov-Smirnov goodness-of-fit statistic D. The closer D is to zero, the better the fit.

3.1.2 Log Normal

The lognormal distribution is expressed by the formula:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-\frac{(ln(x)-\mu)^2}{2\sigma^2}} \tag{5}$$

It is characterized by a "quasi-power-law behaviour" with a final cut-off. Real data cannot have values that tend to infinity and thus represents a good distribution to fit power-law distributed data with a finite-size effect. Different from a power-law, a lognormal distribution does not diverge for small values of the variable and may also fit well the bulk of the distribution. The best fitting for the lognormal distribution computes the coefficient of determination R^2: the closer R^2 is to 1, the better the fit.

4 EXPERIMENT SETUP and METHODOLOGY

4.1 DataSet

This paper analyzes four Java open-source projects: Azureus[2], Jedit[3], Jtopen[4] and Tomcat[5]. Azureus is a P2P file sharing client using the bittorrent protocol; Jedit, as its name suggests, is a programmer's text editor; Jtopen is the open source version of the IBM Toolbox for Java. Finally, Tomcat is an open source software implementation of the Java Servlet and JavaServer Pages technologies.
Relevant statistics related to these systems are reported in Table 1. Generally, there are potentially thousands of classes which belong to each release R_i and a subsequent release R_{i+1}. Only a limited number of parameter based refactorings are undertaken between releases. For example, in Azureus 4.4 - 4.5 there are 7270 classes; among them: 51 have had an add parameter refactoring, 27 have had a remove parameter refactoring. Moreover, in the data that Ref-Finder extracted, a class can have more than one refactoring applied to it. The four Java projects were chosen because they have been actively developed and have several years of development, comprise several releases and because their source code is freely-available from the web.

In general, the software development of OO systems and the scheduling of new version releases may change from system to system, from one platform to another. Some software projects are strongly and rigidly organized, with fixed time delivery of new versions and with a fixed number of sub-versions between the delivery of two versions, with main and patching releases. This is the case with Eclipse or Ubuntu Linux[6]. Some other projects belonging to our dataset are by nature less regular, and

[2]http://sourceforge.net/projects/azureus/
[3]http://sourceforge.net/projects/jedit/
[4]http://sourceforge.net/projects/jt400/
[5]http://archive.apache.org/dist/tomcat/
[6] Eclipse has a main release each year with two patching releases in between. Likewise, Ubuntu has a main release each year, whereas only one patching release in between.

a clear classification in main and patching releases is not feasible. Generally, if R_{i+1} is a patching for R_i, the number of refactored classes may be very small. In such a case, a statistical analysis is meaningless. The best alternative is to analyze changes among two very distant releases, let's say R_i and R_{i+n}, with n large, which presumably includes more refactorings. In this case, the number of classes refactored by an AP or RP may be much larger. For the four systems analyzed, we were faced with similar situations for various systems. For example, we collected the data for 58 different releases of Jtopen, where we had up to ten sub-releases among two main releases. This led us to take only the "main" releases for each project, namely releases labeled with two digits, characterizing relevant steps of process development. In this way, we were able to report only those two releases presenting enough refactored classes, thereby enabling us to perform the statistical analysis.

Table 1 – Projects statistics

Project	Release $R_i \rightarrow R_{i+1}$	# class $\in R_i \bigcap R_{i+1}$	# class with add parameter	# class with remove parameter
Azureus	4.0 - 4.1	6526	49	38
	4.1 - 4.2	6892	0	20
	4.2 - 4.3	6447	120	79
	4.4 - 4.5	7270	51	27
Jedit	3.2 - 4.0	493	35	28
	4.0 - 4.1	586	43	26
	4.1 - 4.2	634	77	70
	4.2 - 4.3	600	100	92
Jtopen	3.0 - 4.0	1702	24	15
	4.0 - 5.0	1871	77	57
	5.0 - 6.0	1936	38	14
	6.0 - 7.0	1963	45	29
Tomcat	6.0 - 6.0.29	1146	46	38
	6.0.29 - 7.0	1111	161	151

4.2 Ref-Finder

Ref-Finder is a tool able to extract up to sixty-three different refactorings — belonging to Fowler's catalog of seventy-two [3] — occurring between two releases. Let us denote by R_i the software release before refactoring and by R_{i+1} the release obtained after refactoring. The tool Ref-Finder provides the name of the classes refactored belonging to R_i and to R_{i+1}. We focused our attention on two refactorings: add parameter and remove parameter, because they are directly able to influence fan-in and fan-out metrics. Ref-Finder was developed by Prete et al. [1] using a template-based refactoring reconstruction approach. The tool's ability to identify refactorings depends on a threshold σ — bounded between a 0 - 1 interval — which defines the similarity level between two source code according to a particular algorithm. Results reported in this paper use σ=0.85; however, we have also used σ=1 obtaining the same results for refactorings analyzed.

5 RESULTS

In this section, we analyze the figures obtained by plotting the experimental CCDF, together with the analytical CCDF obtained by the best fitting procedures. In order to answer our research questions we need to partition the data in different sets. We denote by R_i the software release before refactoring and by R_{i+1} the release obtained after refactoring. In general, when a class is refactored, it is present in both releases, but its fan-in and fan-out change. We need to compare the six sets to understand the effect of refactoring on class coupling. To do this, in our figures we report fan-in and fan-out CCDF's for all these six sets of classes, as well as the best fitting analytical distributions.

For the sake of clarity, we denote these six sets by the following acronyms:

- A_i : All the classes belonging to the release R_i;
- A_{i+1}: All the classes belonging to the release R_{i+1};
- REF_i: The sets of classes refactored by AP, RP with fan in or fan out computed in R_i;
- REF_{i+1}: The sets of classes refactored by AP, RP with fan in or fan out computed in R_{i+1};
- $\overline{REF_i}$: The sets of classes not refactored by AP, RP with fan in or fan out computed in R_i;
- $\overline{REF_{i+1}}$: The sets of classes not refactored by AP, RP with fan in or fan out computed in R_{i+1};

In all figures, black symbols represent fan-in or fan-out of classes after refactoring and white symbols before refactoring. Circles represent data for refactored classes (black circles provide the fan-in or fan-out distribution after refactoring, white circles before refactoring). Squares represent data for non-refactored classes, and diamonds represent data for all a system's classes. Thus, according to the acronyms introduced we have:

- white diamonds: A_i;
- black diamonds: A_{i+1};
- white circle: REF_i;
- black circle: REF_{i+1};
- white squares: $\overline{REF_i}$;
- black squares: $\overline{REF_{i+1}}$;

Finally, the best fitting curves are black dashed lines. Among the various empirical CCDF's there are many overlaps, which may render it difficult to distinguish among the different curves. Such overlaps are unavoidable and an interesting feature of the results. In fact, the overlaps in the figures indicate that different data sets possess the same CCDF, or witness the persistence of statistical distributions after release change. We chose to use the power-law and the lognormal statistical distribution functions for best fitting of the empirical data,since they have already been demonstrated to be the best candidate for fitting the empirical data for fan-in and fan-out distributions in OO software systems [23], [24], [25]. Due to finite size effects, sometimes a true power-law distribution may not appear as a straight line in a log-log plot for large values of the independent variable as well as for small values, as it should be for an ideal, infinite size sample. On the contrary, when a variable is lognormally distributed, a combination of finite size effects and parameter values may provide an almost linear shape in a defined range of a log-log plot. Thus using a power-law for best fitting purposes may provide a very good fitting even for a true lognormal distribution. Conversely, due to the finite size effects, a lognormal can provide a very good best fit for a true power-law distribution. Due to these possibilities, we decided to apply both kinds of best fitting functions for fan-in and fan-out data. Our best fitting results, reported in Tables 2 and 3, confirm that fan-in is well modelled by a power-law distribution, while a lognormal distribution is applicable for fan-out, for all the six sets analyzed. Thus, in the figures we report only the power-law best fitting CCDF's for fan-in and the lognormal best fitting CCDF's for fan-out.

Figures 1-8 display the empirical fan-in and fan-out distributions for Azureus together with the best fitting curves, reproduce the typical situation for all the systems analyzed. Such figures show the results for a single refactoring: from Azureus 4.0 - 4.1 (Figure 1), for Azureus 4.1 - 4.2 (Figure 2), for Azureus 4.2 - 4.3 (Figure 3), for Azureus 4.4 - 4.5 (Figure 4) for the fanin CCDF. In the same way Figures 1-8 show the result of a single refactoring for fanout distribution CCDF. The CCDF for refactored classes (sets REF_{i+1} and REF_i) display a vertical shift with respect to the CCDF of all system's classes, while the CCDF for not refactored (sets $\overline{REF_{i+1}}$ and $\overline{REF_i}$) classes almost completely overlap the latter ($A_{i,i+1}$). This second point is due to the fact that, on average, the refactored classes are a small fraction of the total, so the set of non-refactored classes (according to AP and RP refactorings) practically coincides with the set of all system's classes. This occurs regardless of evaluating fan-in distributions before or after refactoring and the same holds for fan-out distributions. This vertical shift is an experimental result, and indicates that developers tend to refactor classes with higher fan-in or fan-out on average, as will be discussed in the next section.

Figures 1 to 4 suggest that power-law distributions hold for fan-in for the sets $A_{i,i+1}$ and $\overline{REF_{i,i+1}}$. Furthermore, they show that the same power-law holds for the sets of refactored classes before and after refactoring. Similar considerations may be applied to fan-out empirical distributions. Figures 5 to 8 suggest that lognormal distribution functions may be used for sets $A_{i,i+1}$ and $\overline{REF_{i,i+1}}$. Furthermore, they show that the same lognormal holds for the sets of refactored classes before and after refactoring. The considerations above are supported by the results obtained by the best fitting procedures.

Consider first the results obtained for fan-in using the power-law distribution for best fitting, reported in Tab. 2. Apart from a few cases, the goodness of fit parameters indicate that the power-law distribution can be a good model for the fan-in statistical distribution of refactored classes, both before and after refactoring. This property, to authors knowledge, has never been reported in literature. In fact, even if the power law has been demonstrated to be a good model for fan-in distributions of the entire system [25] [9], this has never been found for sets REF_i and REF_{i+1}.

Consider next the results obtained for fan-out using the lognormal distributions, reported in Table 3. Apart from a very few cases, the lognormal distribution fits very well the fan-out curves for refactored classes, both before and after refactoring.

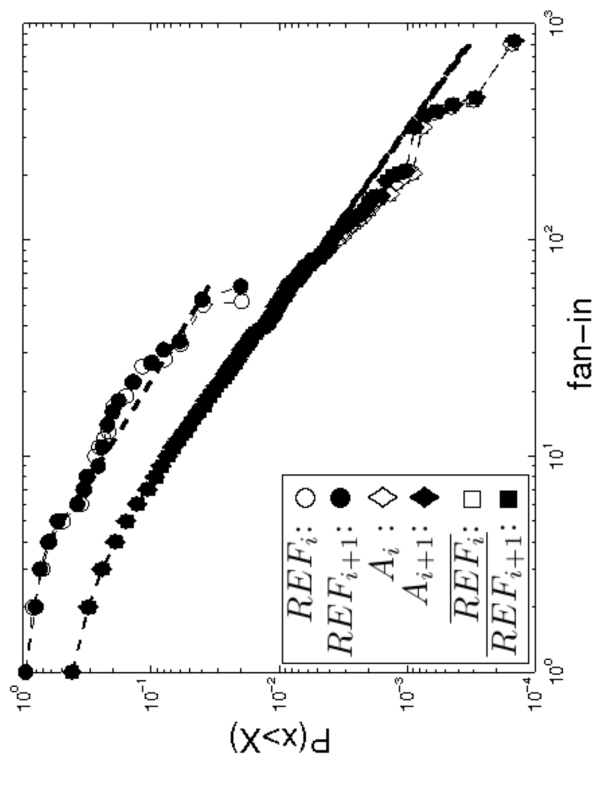


Figure 1 – The CCDF of fan-in for refactorig from Azureus 4.0 (R_i) to - Azureus 4.1(R_{i+1})

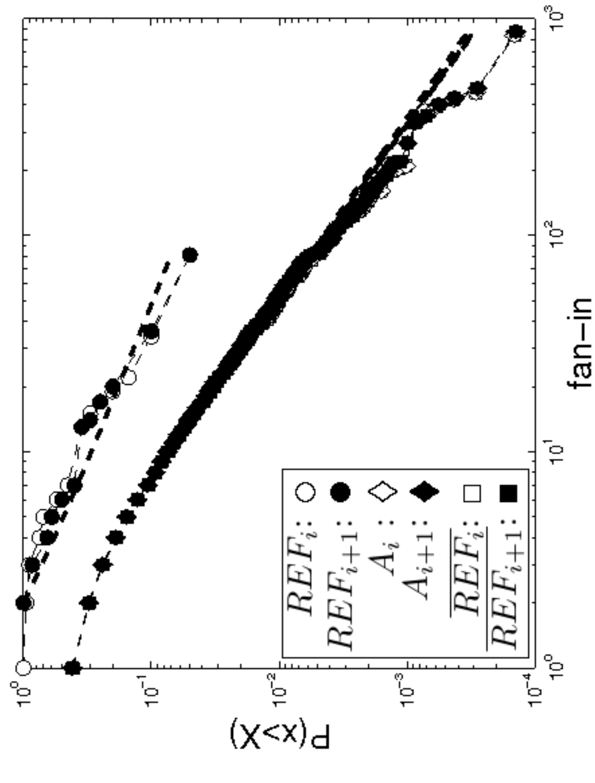


Figure 2 – The CCDF of fan-in for refactorig from Azureus 4.1 (R_i) to - Azureus 4.2(R_{i+1})

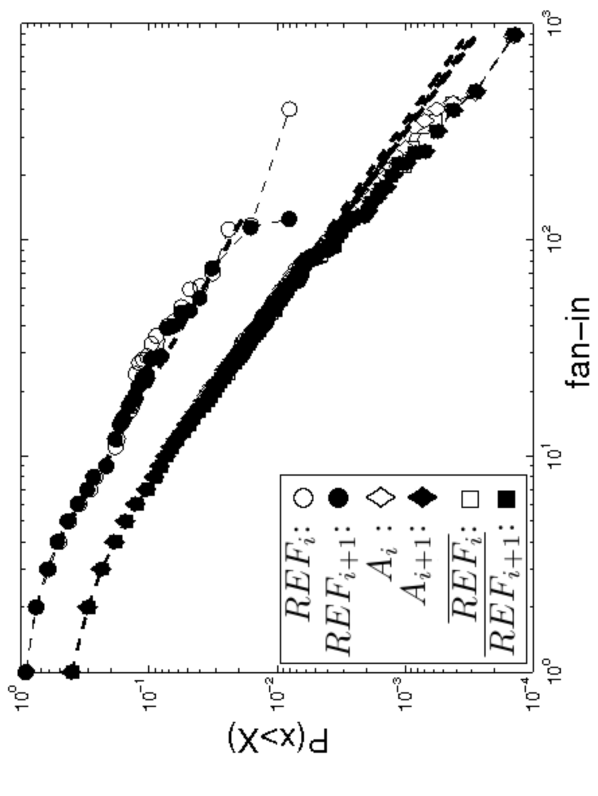


Figure 3 – The CCDF of fan-in for refactorig from Azureus 4.2 (R_i) to - Azureus 4.3(R_{i+1})

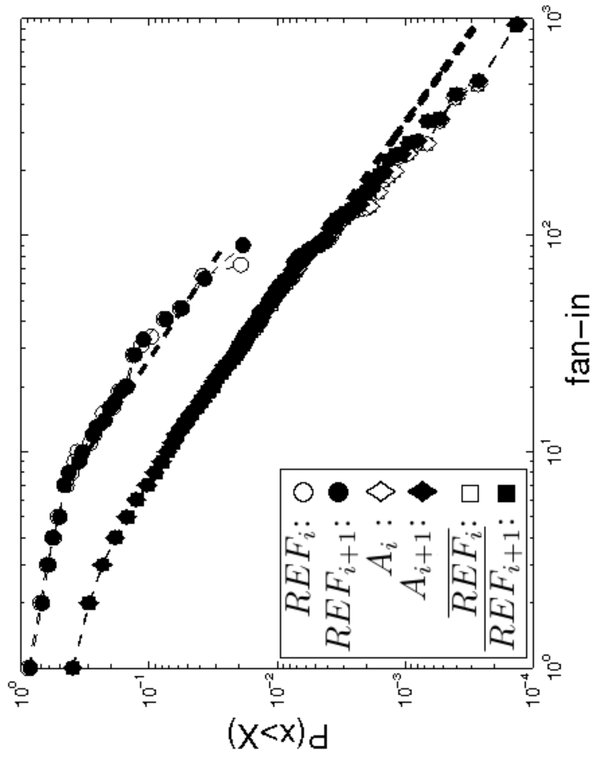


Figure 4 – The CCDF of fan-in for refactorig from Azureus 4.3 (R_i) to - Azureus 4.5(R_{i+1})

Table 2 – Power-law fitting statistics for the **fan-in** of the four systems. In the releases column, AR and BR stand for "after refactoring" and "before refactoring" respectively. The parameters α, x_0, and D have been computed on the refactored classes (R), on the not-refactored classes (NR) and on all the system classes (all).

Releases	α_R	x_{0R}	D_R	α_{NR}	x_{0NR}	D_{NR}	α_{all}	x_{0all}	D_{all}
JTOpen									
3.0-4.0 AR	2.185	3	0.099	1.872	3	0.041	1.878	3	0.039
3.0-4.0 BR	3.450	4	0.073	1.875	3	0.042	1.887	3	0.043
4.0-5.0 AR	1.644	2	0.091	1.902	3	0.041	1.890	3	0.039
4.0-5.0 BR	1.870	7	0.087	1.891	3	0.039	1.878	3	0.039
5.0-6.0 AR	1.877	4	0.075	1.904	3	0.037	1.897	3	0.036
5.0-6.0 BR	1.910	4	0.098	1.901	3	0.036	1.894	3	0.039
6.0-7.0 AR	1.785	3	0.069	1.898	3	0.039	1.893	3	0.038
6.0-7.0 BR	1.830	3	0.080	1.897	3	0.037	1.894	3	0.036
JEdit									
3.2-4.0 AR	1.830	2	0.120	2.135	4	0.040	2.060	4	0.042
3.2-4.0 BR	1.660	4	0.123	2.162	8	0.048	2.065	7	0.051
4.0-4.1 AR	1.798	3	0.072	2.227	4	0.035	2.101	4	0.029
4.0-4.1 BR	1.810	18	0.075	2.122	7	0.054	2.060	7	0.042
4.1-4.2 AR	1.973	4	0.055	2.402	4	0.051	2.093	4	0.052
4.1-4.2 BR	1.970	16	0.066	2.220	6	0.045	2.119	6	0.034
4.2-4.3 AR	1.868	4	0.075	2.078	9	0.043	1.961	4	0.045
4.2-4.3 BR	2.110	7	0.046	2.323	10	0.061	2.086	9	0.050
Azureus									
4.0-4.1 AR	2.017	4	0.091	2.226	11	0.023	2.226	11	0.026
4.0-4.1 BR	2.03	4	0.109	2.218	10	0.021	2.219	10	0.024
4.1-4.2 AR	1.673	2	0.133	2.196	10	0.026	2.229	12	0.027
4.1-4.2 BR	2.6	13	0.105	2.225	11	0.025	2.226	11	0.026
4.2-4.3 AR	1.933	4	0.042	2.227	10	0.024	2.244	11	0.026
4.2-4.3 BR	1.8	3	0.069	2.22	10	0.023	2.2	11	0.027
4.4-4.5 AR	2.099	7	0.075	2.218	12	0.020	2.22	12	0.020
4.4-4.5 BR	2.22	9	0.111	2.226	12	0.023	2.226	12	0.026
Tomcat									
6.0-6.029 AR	2.150	2	0.090	2.670	14	0.041	2.710	14	0.040
6.0-6.029 BR	2.160	2	0.081	2.680	14	0.036	2.670	12	0.029
6.029-70 AR	1.720	2	0.083	2.060	3	0.049	2.020	3	0.050
6.029-70 BR	1.600	1	0.067	2.890	14	0.045	2.670	14	0.048

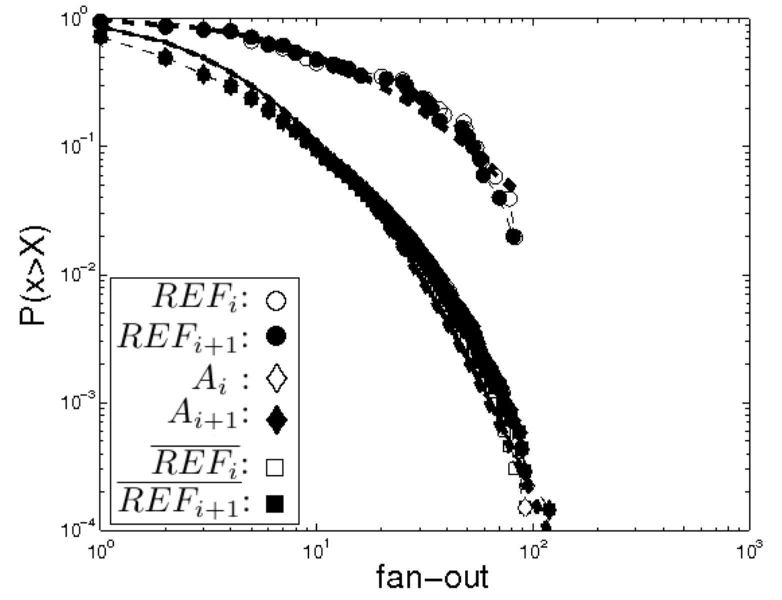


Figure 5 – The CCDF of fan-out for refactorig from Azureus 4.0 (R_i) to - Azureus 4.1(R_{i+1})

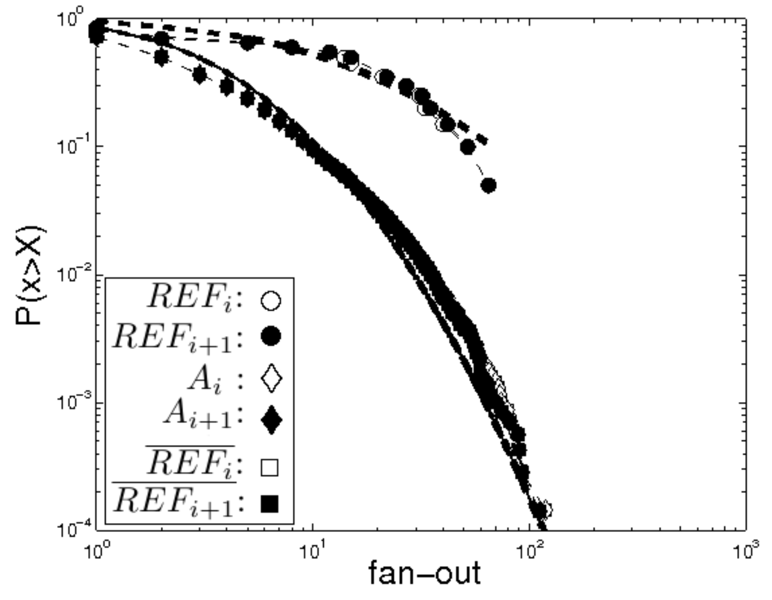


Figure 6 – The CCDF of fan-out for refactorig from Azureus 4.1 (R_i) to - Azureus 4.2(R_{i+1})

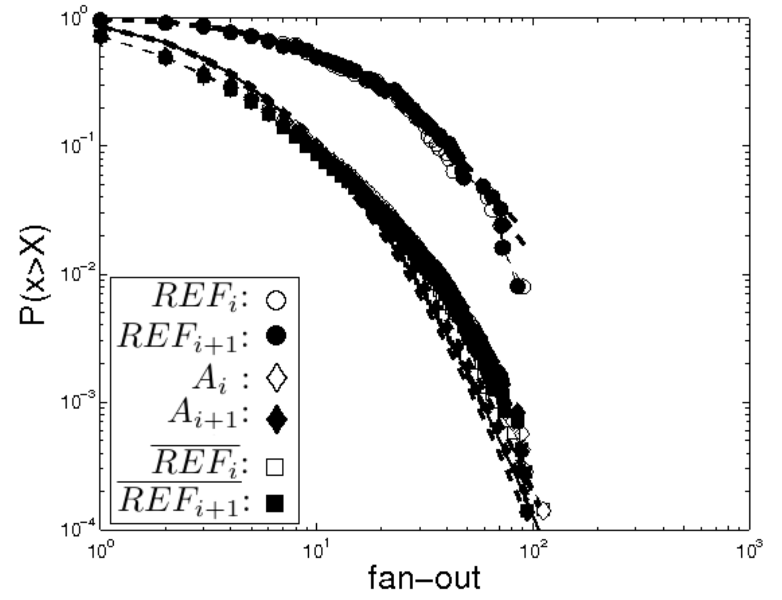


Figure 7 – The CCDF of fan-out for refactorig from Azureus 4.2 (R_i) to - Azureus 4.3(R_{i+1})

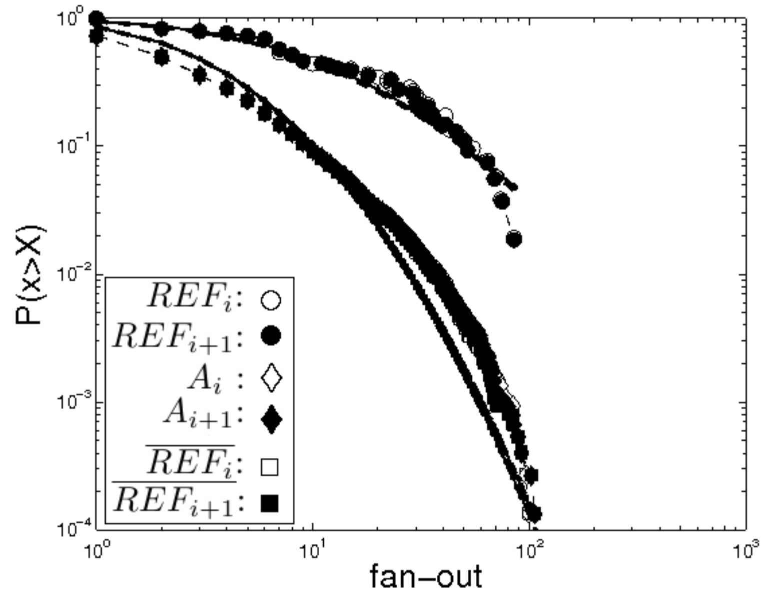


Figure 8 – The CCDF of fan-out for refactorig from Azureus 4.4 (R_i) to - Azureus 4.5(R_{i+1})

Table 3 – Lognormal fitting statistics for **fan-out** of the four systems analyzed. In the releases column, AR and BR stand for "after refactoring" and "before refactoring" respectively. The parameters R^2, μ, and σ have been computed on the refactored classes (R), on the not-refactored classes (NR) and on all the system classes (all).

Releases	R^2_R	μ_R	σ_R	R^2_{NR}	μ_{NR}	σ_{NR}	R^2_{all}	μ_{all}	σ_{all}
JTOpen									
3.0-4.0 AR	0.966	2.146	0.727	0.984	1.327	0.968	0.984	1.340	0.970
3.0-4.0 BR	0.949	2.008	0.617	0.984	1.309	0.958	0.985	1.322	0.958
4.0-5.0 AR	0.979	1.828	0.791	0.984	1.324	0.977	0.984	1.347	0.975
4.0-5.0 BR	0.978	1.749	0.821	0.984	1.320	0.972	0.984	1.339	0.970
5.0-6.0 AR	0.940	1.857	1.323	0.984	1.336	0.963	0.985	1.348	0.975
5.0-6.0 BR	0.936	1.837	1.349	0.984	1.334	0.962	0.985	1.345	0.974
6.0-7.0 AR	0.983	2.272	0.916	0.984	1.338	0.968	0.985	1.362	0.978
6.0-7.0 BR	0.983	2.248	0.856	0.984	1.324	0.967	0.985	1.349	0.975
JEdit									
3.2-4.0 AR	0.952	1.646	0.893	0.972	0.920	0.787	0.979	0.990	0.825
3.2-4.0 BR	0.950	1.603	0.924	0.972	0.934	0.791	0.978	1.000	0.828
4.0-4.1 AR	0.983	1.958	0.708	0.969	0.807	0.734	0.979	1.002	0.847
4.0-4.1 BR	0.972	1.850	0.772	0.964	0.783	0.695	0.979	0.990	0.825
4.1-4.2 AR	0.977	1.610	1.109	0.972	0.918	0.808	0.977	0.976	0.857
4.1-4.2 BR	0.973	1.586	1.038	0.975	0.946	0.807	0.979	1.002	0.847
4.2-4.3 AR	0.980	1.678	0.938	0.973	0.899	0.811	0.977	0.966	0.849
4.2-4.3 BR	0.979	1.650	0.968	0.972	0.908	0.816	0.977	0.976	0.857
Azureus									
4.0-4.1 AR	0.977	2.309	1.228	0.983	1.086	0.981	0.983	1.098	0.991
4.0-4.1 BR	0.975	2.342	1.247	0.983	1.086	0.977	0.983	1.099	0.988
4.1-4.2 AR	0.899	2.368	1.421	0.983	1.085	0.983	0.983	1.089	0.987
4.1-4.2 BR	0.871	2.457	1.388	0.983	1.092	0.985	0.983	1.096	0.990
4.2-4.3 AR	0.989	2.291	1.095	0.981	1.032	0.959	0.981	1.061	0.980
4.2-4.3 BR	0.989	2.311	1.038	0.982	1.062	0.970	0.983	1.091	0.989
4.4-4.5 AR	0.975	2.154	1.351	0.981	1.047	0.975	0.981	1.058	0.986
4.4-4.5 BR	0.964	2.203	1.350	0.981	1.050	0.974	0.981	1.061	0.9
Tomcat									
6.0-6.029 AR	0.920	1.975	1.012	0.980	1.117	0.951	0.980	1.156	0.970
6.0-6.029 BR	0.927	1.901	1.059	0.982	1.139	0.964	0.982	1.176	0.982
6.029-70 AR	0.977	1.610	1.046	0.982	1.103	0.921	0.984	1.178	0.957
6.029-70 BR	0.976	1.575	1.017	0.977	1.075	0.941	0.980	1.154	0.970

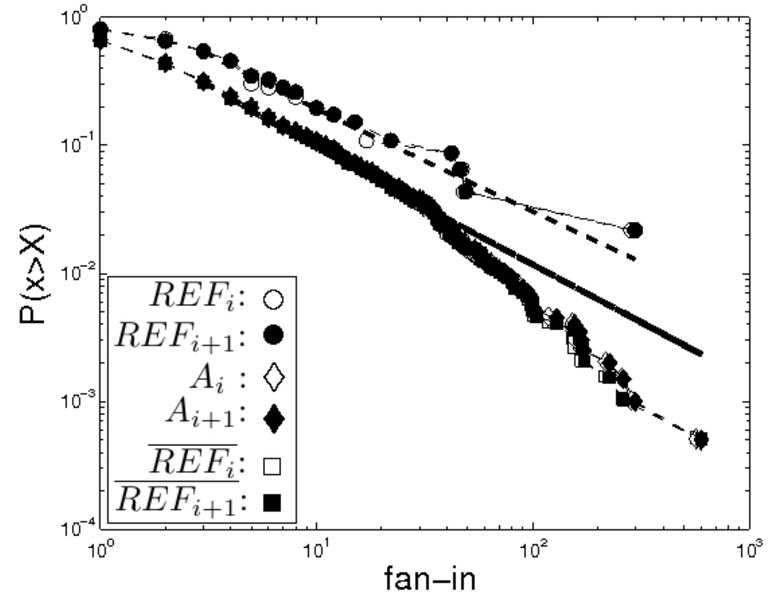


Figure 9 – The CCDF of fan-in for refactorig from Jtopen 6.0 (R_i) to - Jtopen 7.0(R_{i+1})

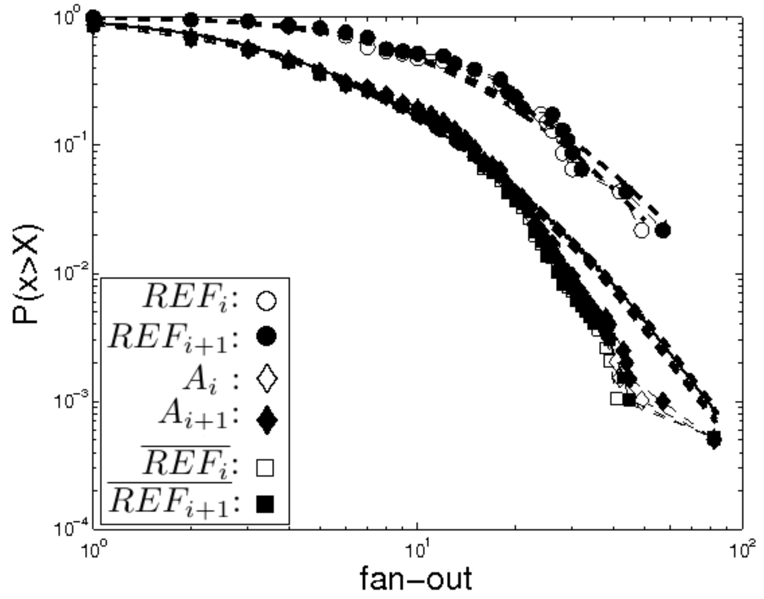


Figure 10 – The CCDF of fan-out for refactorig from Jtopen 6.0 (R_i) to - Jtopen 7.0(R_{i+1})

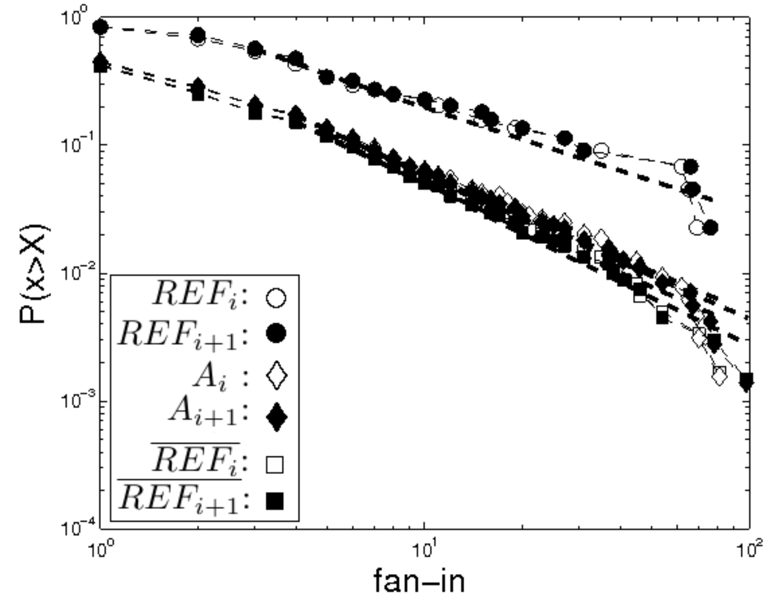


Figure 11 – The CCDF of fan-in for refactorig from Jedit 4.0 (R_i) to - Jedit 4.1(R_{i+1})

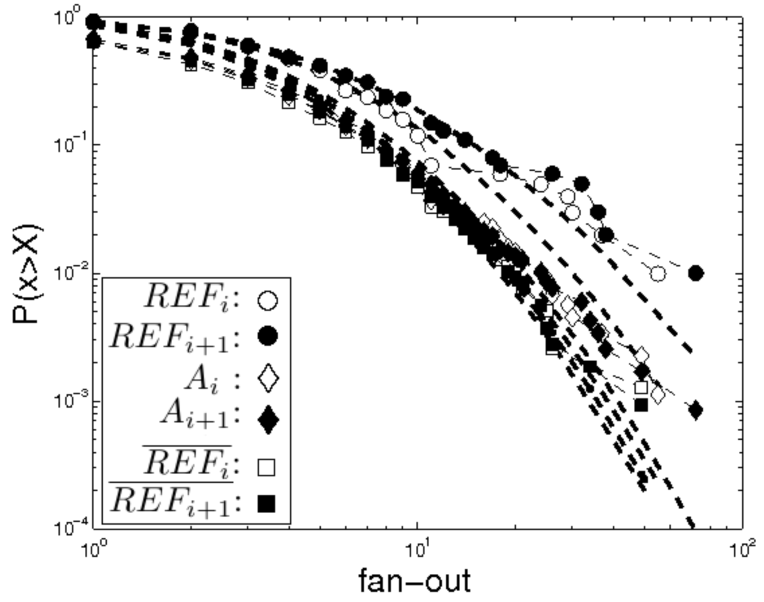


Figure 12 – The CCDF of fan-out for refactorig from Jedit 4.2 (R_i) to - Jedit 4.3(R_{i+1})

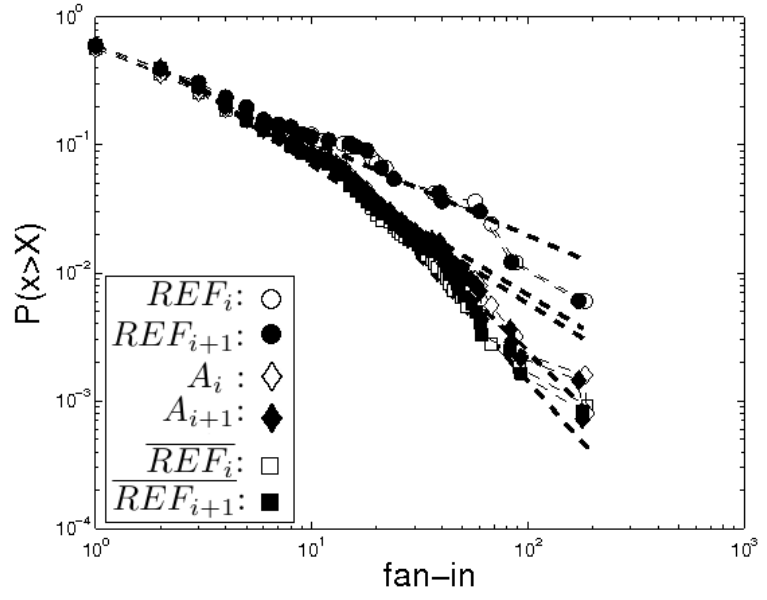


Figure 13 – The CCDF of fan-in for refactorig from Tomcat 6.0.29 (R_i) to - Tomcat 7.0(R_{i+1})

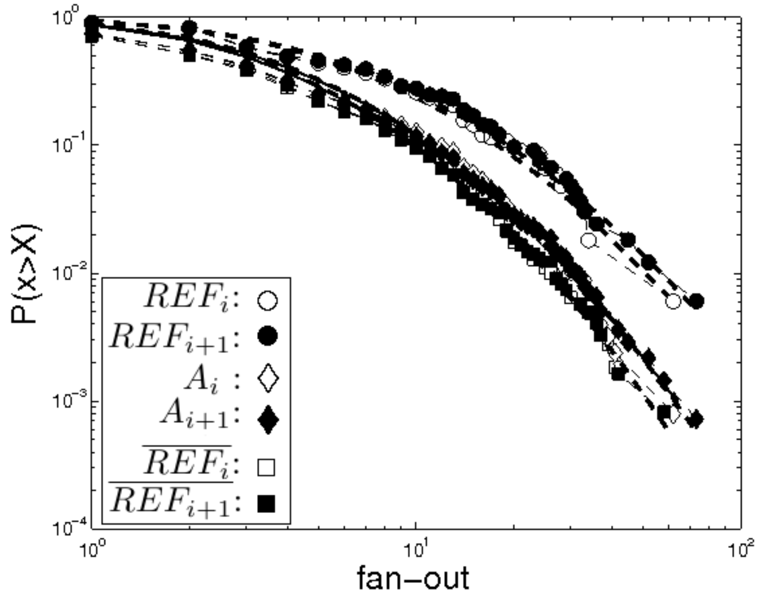


Figure 14 – The CCDF of fan-out for refactorig from Tomcat 6.0.29 (R_i) to - Tomcat 7.0(R_{i+1})

Once again, this result has not been reported in the literature, since the ability of a lognormal to model fan-out empirical distributions has been investigated only for the entire system.

6 DISCUSSION

In this section, we discuss in detail the results presented, their possible consequences and provide the answers to the original research questions. The analysis of empirical and best fitting CCDF's shows that the curves for sets REF_i and REF_{i+1} display a general vertical shift with respect to the other sets and are also above the others. Namely, fan-in and fan-out for refactored classes are, on average, larger than those of non-refactored classes, both before and after being refactored. This property appears applicable to all sets by the examination of the empirical CCDF's. In fact, for each fixed value $\bar{x}$ in the horizontal axes of the plots, the higher the corresponding Y, the higher is the percentage of classes with fan-in/fan-out larger than $\bar{x}$. This property is confirmed by the parameters obtained through the best fitting analysis.

In the case of fan-in best fitting, the power-law best fitting exponent α (α_R in Table 2) is generally smaller than the corresponding power-law best fitting exponents for non-refactored classes and for the A_i set (α_{NR} and α_{all} in Table 2). This is in agreement with Figures 1-4, where the CCDF's for the sets REF_i and REF_{i+1} are above the others and indicate that average values of fan-in for refactored classes, before and after refactoring, are larger than in the other sets.

In the case of fan-out best fitting, the log-normal best fitting parameter μ for refactored classes, both before and after refactoring (μ_R in Table 3), is generally larger than the corresponding parameters for the sets of non-refactored classes and and for the A_i set (μ_{NR} and μ_{all} in Tab. 3). In the log-normal distribution function, the mean and median statistics are $e^{\mu+\frac{\sigma^2}{2}}$ and e^{μ}, respectively. Thus, our best fitting analysis confirms that the sets REF_i and REF_{i+1} have, on average, larger values for fan-out.

The vertical shift is an experimental result and from a refactoring point of view provides a clear indication about the way developers work when applying AP and RP refactorings to a system's classes: they tend to select classes with higher fan-in or fan-out values on average. This indicates that AP and RP activities are not applied to classes at random and that some other criteria must be at play. In fact, if the choice was random, namely if the classes for applying AP and RP refactorings were extracted with a uniform random probability, the statistical distribution would not change and would not display any vertical shift, especially before refactoring.

We now concentrate on fan-in. The empirical results and the best fitting analysis provide another interesting result. They suggest that the same statistical distribution functions may be used for modelling refactored classes, with a different set of parameters and that the distribution function remains approximately the same after refactoring.

It is already known in the literature [23], that fan-in is power-law distributed across all system's classes. Our studies confirm power-law distributions for fan-in of all system's classes and for the overlapping set of non-refactored classes; they also suggest that the same distribution can still hold for the set of refactored classes, both before and after refactoring, indicating a persistence of such a distribution in the set of classes. This is not a trivial result, since the effect (and maybe the original purpose) of applying AP and RP refactoring is exactly to change the fan-in of the relevant class, breaking an existing coupling with a linked class or introducing a new coupling with another class.

There is no guarantee that the same distribution must hold for refactored classes alone; namely, there is no indication that for the same model to be applicable to the subset of refactored classes. In principle, developers can choose *ad hoc* to refactor a particular subset of classes so that the fan-in distribution for such a subset is not a power-law; trivially, one can just choose classes with exactly the same fan-in, obtaining a flat distribution. Examining the best fitting parameters values, the same power-law is able to model fan-in distribution before and after refactoring, since the parameter values do not change for such sets. In particular, the power-law exponent α provides the same slope in the log-log plot for both sets and the lower cut-off x_0 indicates that, for both distributions, the power-law behaviour starts from the same point, usually at the very beginning of the distribution (for almost the entire range of fan-in values). In contrast, the slope α changes with respect to the non-refactored classes and to the entire system's classes. In particular, whenever the fit is good, the value of α decreases for refactored classes. This indicates that the straight line in the log-log plot has a smaller slope (α appears with a minus sign in the power-law function 4). Thus, the best fitting parameters confirm the tendency of developers to select classes with relatively high fan-in for refactoring, since the smaller the module of the power-law exponent, the more the power-law is vertically shifted, especially in the tail of the distribution.

The overlap of fan-in distributions before and after refactoring indicates a re-organization or a redistribution of fan-in across the classes of the refactored set in a way which preserves the same overall statistical distribution, while fan-in of each single class eventually changes. The situation appears similar for fan-out empirical results, with the log-normal distribution function in place of the power-law. Previous work has shown that the best fitting analytical distribution functions for fan-out of the overall software system are lognormal [24] [25]. Our results suggest that lognormal distribution functions may also be used to model refactored classes, but with different values of the best fitting parameters. The refactored classes possess the same empirical distribution for fan-out before and after refactoring, indicating a persistence of such a distribution, even if fan-out of each single class has been changed by the application of AP and RP refactorings; this eventually causes each class to change the group of classes it is coupled to. Additionally, a re-organization or redistribution of fan-out across the classes occurs in such a way that the same overall statistical distribution is preserved, while fan-out of each single class eventually changes.

We previously stated for fan-in that the vertical shift indicates that classes to be refactored are not randomly chosen among all a system's classes. This is also confirmed for the fan-out data. As a consequence, developers apply AP and RP activities in such a way that the lognormal fan-out distribution is still preserved in the limit of our best fitting confidence. Furthermore, as already discussed, such a lognormal distribution remains the same for the distributions before and after refactoring, confirmed by comparing the best fitting parameters values for each couple of sets. This means that, even if removing parameters to methods or adding parameters to methods changes the class coupling, fan-out preserves the same statistical distribution, irrespective of how single class coupling is modified. On the contrary, comparing the same best fitting parameters with those obtained for the distributions of non-refactored classes and of all the system's classes, they show a net increment of the mean parameter (μ), confirming the tendency of developers to refactor classes with a relatively high fan-out. The projects we have analyzed do not provide enough information related to how, when and why refactoring is performed. The tool we used to extract refactorings, Ref-Finder, is able only to identify and classify AP and RP refactorings occurring between different releases. So we can only speculate about the reasons why AP and RP refactorings are not able to change coupling significantly. We believe there were non-refactoring-driven releases during the evolution of our projects. For this reason, between two release R_i and R_{i+1} there are refactoring driven activities and significant maintenance activities; the influence of the latter on refactored classes might reduce the refactoring benefits on software coupling. Finally, we have to consider that refactoring does not strictly require a reduction of software coupling. Thus, AP and RP refactorings can achieve other improvements on software which do not change coupling.

On the basis of these results we can now answer the research questions.

- RQ1- Are there analytical distribution functions able to describe fan-in in refactored and non-refactored classes? Are there differences among refactored and non-refactored classes?
  We found that fan-in empirical distributions are in general well modelled by a power-law for both refactored and not-refactored classes. In general, best fitting provides better results for non-refactored classes, but this has to be ascribed to the differences in the populations of the two sets, given that refactored classes are only a small part of the total classes and the fact that a power-law fit improves when the set is larger. Furthermore, we found significant differences in the values of the best fitting parameters, indicating that developers focus their attention on refactoring classes with relatively high fan-in.
- RQ2- Does refactoring change fan-in coupling overall?
  The best fitting parameters show a negligible change in their values in almost all the analyzed cases when computed before or after refactoring. This indicates that the two types of refactorings analyzed have an overall null effect on the fan-in class coupling, even if each single class eventually changes the classes it is coupled to. This result should take into account all those cases in which refactoring is planned with the single purpose of reducing coupling to decrease maintenance effort as well as for other reasons.
- RQ3- Are there analytical distribution functions able to describe fan-out in refactored and non-refactored class? Are there differences among refactored and non-refactored classes? Again, the same considerations just described apply for the fan-in case. The empirical distributions of fan-out are well modelled by a lognormal distribution, for both refactored and non-refactored classes. In this case, the best fitting quality presents no appreciable differences among the two types, since the R^2 coefficient of determination is less sensitive to differences in the two populations. Here again there are differences in the best fitting parameters values showing that the classes selected for refactoring have relatively high fan-out.
- RQ4- Does refactoring change fan-out coupling overall?
  The empirical CCDF's and the parameter values show that no appreciable changes occur in the distribution of fan-out before and after refactoring. So the two types of refactorings analyzed have an overall null effect on fan-out class coupling, even if each single class eventually changes the classes it is coupled to. Once again, this must be taken into account when refactoring is planned explictly to reduce coupling.

7 THREATS TO VALIDITY

Threats to internal validity concern confounding factors which could influence the values of fan-in and fan-out without being related to refactoring. In fact, fan-in and fan-out metrics can be influenced by different maintenances activities, such as feature introduction, enhancement request and bug fixing. A key threat to the internal validity of the study presented is therefore whether significant changes were made to classes (which changed the fan-in or fan-out values) which are not related to parameter-based refactorings identified in this study. Put another way, there is the possibility that other maintenance activity on classes also altered the fan-in and fan-out values - but were not included in our analysis. Refactored and non-refactored classes, as we have defined them, might have undergone significant modification that changed the fan-in and fan-out. In defence of this threat, while it is possible of course for coupling to be introduced into, and removed from a class in many ways, we observed a considerable proportion of the addition and removal of coupling to arise through changes that either added a parameter or removed a parameter [31]. Moreover, it is also possible that these refactorings would be the only maintenance activities performed upon many refactored classes and, equally, that many changes made to classes (and not covered by our refactorings) do not influence fan-in and fan-out anyway. In addition, our study takes into account four projects, across different releases, with several hundreds and sometimes thousands of classes. Thus, the "noise" effect which might be given by other maintenance operations, albeit present, should not invalidate our conclusions. In fact, our analysis takes into account the possibility that an add parameter would nullify a remove parameter refactoring (and *vice versa*) thereby cancelling the expected change in the distributions before and after refactoring.

Threats to external validity concern the degree to which we can draw general conclusions from our results. Refactoring operations (i.e., why refactoring is done) depends to a large degree on a potentially large number of relevant variables such as developer experience, the amount of time that a developer has to carry out refactoring and the extent to which classes are problematic. Our case study is composed of open-source systems written in Java and represents a small sample of software systems. Even if we focused our analysis on general findings, largely ignoring any comments dealing with specific cases, it would be useful to analyze more systems - especially if developed in industry environments - to confirm or reject our findings. We have considered only parameter-based refactorings, but we need to consider the possibility that there are many other types of refactoring that a developer might apply to their code which influence the value of fan-in and fan-out. For this reason, it would be useful to explore more refactorings in the same spirit as we have herein. Threats to construct validity focus on how many refactorings are accurately detected and how correct the identified refactorings are; this depends largely on the reliability of Ref-Finder [1]. The authors went to great lengths through sampling and checks to ensure that the output of the Ref-Finder tool was as expected and we are confident that the data extracted using the tool is an accurate representation of the systems.

8 CONCLUSION AND FUTURE WORK

This paper investigated the relationship between parameter-based refactorings and fan-in, fan-out metrics using multiple releases of four Java open-source systems. Using the tool Ref-Finder, we identified the *add parameter* and *remove parameter* refactorings performed between a release R_i and its subsequent release R_{i+1}. Our results indicate that refactored and non-refactored classes have different statistical distributions for fan-in and fan-out metrics, in the sense that while the statistical distribution best fitting functions are the same, the best fitting parameters differ systematically, causing a vertical shift in the CCDF's of the metrics for refactored classes. Average fan-in and fan-out values are always larger for refactored classes. In other words values of fan-in and fan-out are larger in refactored classes with respect to non-refactored classes. This was a surprising result from the analysis. We might expect developers to prefer the "safer" option of modifying classes where the impact of making a change is less likely to cause side-effects, i.e. by refactoring classes with a high fan-out and low-fan-in; this is based on the caveat that modifying classes with a high fan-in could impact many classes if functionality that they depend on is touched in any way. That said, it is not always a choice that the developer has. A refactoring might be an urgent response to a problematic piece of code.

Our results also show that fan-in in refactored and non-refactored class can be modelled by a power-law, whereas fan-out is modelled by a lognormal distribution. Parameter-based refactorings do not exhibit any impact on fan-in and fan-out values of software system; their evolution does not influence the "shape" of such distributions. We conjecture that *add parameter* and *remove parameter* interactions, have only a negligible net effect. If reduced coupling is an aim of refactoring, then careful consideration should be given to which refactorings are chosen and how they might influence the overall values of fan-in and fan-out. The empirical study of refactoring presented suggest that we should evaluate both how refactoring modifies the software "qualitatively", and how it influences software "quantitatively".
Generally, it is assumed that any increase in coupling is generally conceived as a bad thing. However, we need to be mindful of the possibility that different forms of coupling might have different effects on the maintainability of a system. Relatively large amounts of one form of coupling might be preferable to small amounts of another form. Refactoring might introduce a harmful form of coupling while eliminating a less damaging type. Of course, this hypothesis would be if we know *a priori* which types of coupling are harmful and which are not.
In future work, we intend to investigate more refactorings not listed in this paper (and not necessarily those in the seventy-two proposed by Fowler) but yet able to influence fan-in and fan-out. We intend to explore the influence and effect that the possible confounding factors, such as regular non-refactoring maintenance activity, might have on the results presented; this will address the possibility that coupling could be affected by other non-add/non-remove parameter activities. Finally, while fan-in and fan-out give a finer-grained profile of coupling than many existing coupling metrics, it would be useful to explore the composition of fan-in and fan-out to determine the exact influence on distributions of the different coupling forms.

References

[1] K. Prete, N. Rachatasumrit, N. Sudan and M. Kim, Template-based Reconstruction of Complex Refactorings. Proceedings of the 26th IEEE International Conference on Software Maintenance, 2010. DOI:10.1109/ICSM.2010.5609577

[2] V. Basili, G. Caldiera, and D. H. Rombach, The Goal Question Metric Paradigm Encyclopedia of Software Engineering. John Wiley and Sons, 1994.

[3] M. Fowler, Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 1999. ISBN: 0-201-48567-2

[4] L. Briand, J. Daly, and J. Wust, A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering, vol. 25, pp. 91-121, 1999. DOI:10.1109/32.748920

[5] S. Henry, and D. Kafura, Software Structure Metrics Based on Information Flow. IEEE Transactions on Software Engineering, vol., Issue 5, pp. 510 - 518, 1981. DOI:10.1109/TSE.1981.231113

[6] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design. IEEE Transactions on Software Engineering, vol. 20, pp.476-493, 1994. DOI:10.1109/32.295895

[7] M.E.J. Newman, Power-laws, Pareto, distributions and Zipf's law. Contemporary Physics, Vol. 46, pp. 323-351, 2005. DOI:10.1080/00107510500052444

[8] A. Clauset, C.R. Shalizi and M. Newman, Power-Law Distributions in Empirical Data. SIAM Review, v.51 n.4, p.661-703, 2009. DOI:10.1137/070710111

[9] K. Stroggylos and D. Spinellis, Refactoring–Does It Improve Software Quality?. ICSE International Workshop on Software Quality, International, 2007. DOI:10.1109/WOSQ.2007.11

[10] S. Counsell, Y. Hassoun, G. Loizou, R. Najjar, Common refactorings, a dependency graph and some code smells: an empirical study of Java OSS. International Symposium on Empirical Software Engineering, pp. 288-296, 2006. DOI:10.1145/1159733.1159777

[11] S. Demeyer, S. Ducasse and O. Nierstrasz, Finding Refactorings via Change Metrics. Proceedings of International Conference on Object-oriented Programming Systems, Languages and Applications, pp. 166-177, 2000. DOI:10.1145/353171.353183

[12] A. Mubarak, S. Counsell and R. M. Hierons, An Empirical Study of "Removed" Classes in Java Open-Source Systems. Advanced Techniques in Computing Sciences and Software Engineering, pp. 99-104, 2008. DOI:10.1007/978-90-481-3660-5_17

[13] A. Mubarak, S. Counsell and R. M. Hierons, An Evolutionary Study of Fan-in and Fan-out Metrics in OSS. IEEE Conference on Research Challenges in Information Science, pp. 473-482, 2010. DOI:10.1109/RCIS.2010.5507329

[14] W. Opdyke, Refactoring Object-Oriented Frameworks. PhD thesis, Univ. Illinois at Urbana-Champaign, 1992.

[15] L. Zhao and J. Hayes, Predicting Classes in Need of Refactoring: An Application of Static Metrics. 2nd International PROMISE Workshop, 2006.

[16] D. Advani, Y. Hassoun and S. Counsell, Extracting refactoring trends from open-source software and a possible solution to the related refactoring conundrum. Proceedings of ACM Symposium on Applied Computing, 2006. DOI:10.1145/1141277.1141685

[17] L. Tokuda and D. Batory, Evolving Object-Oriented Designs with Refactorings. Automated Software Engineering, vol. 8, pp.89-120, 2001. DOI:10.1023/A:1008715808855

[18] R. Tonelli, Y.-C. Lai and C. Grebogi, Feedback synchronization using pole-placement control. International Journal of Bifurcation and Chaos in Applied Sciences and Engineering 10 (11) , pp. 2611-2617, 2000. DOI:10.1142/S0218127400001675

[19] S. Counsell, R. M. Hierons, R. Najjar, G. Loizou, Y. Hassoun, The Effectiveness of Refactoring, Based on a Compatibility Testing Taxonomy and a Dependency Graph. IEEE Testing: Academia and Industry Conference - Practice And Research Techniques, pp. 181-192, 2006. DOI: 10.1109/TAIC-PART.2006.33

[20] D. Binkley, M. Ceccato, M. Harman, F. Ricca, and P. Tonella, Automated Refactoring of Object Oriented Code into Aspects. In Proceedings of International Conference on Software Maintenance, pp. 27-36, 2005. DOI:10.1109/ICSM.2005.27

[21] T. Mens and T. Tourw, A Survey of Software Refactoring. IEEE Transactions on Software Engineering, vol. 30, pp. 126-139, 2004. DOI:10.1109/TSE.2004.1265817

[22] L. Briand, P. Devanbu, and W. Melo, An investigation into coupling measures for C++, 19th International Conference on Software Engineering, pp. 412-421, 1997. DOI:10.1109/ICSE.1997.610307

[23] P. Louridas, D. Spinellis and V. Vlachos, Power-laws in software. ACM Transactions on Software Engineering and Methodology. vol. 18, art. 2, 2008.DOI:10.1145/1391984.1391986

[24] G. Concas, M. Marchesi, A. Murgia and R. Tonelli, An Empirical Study of Social Networks Metrics in Object Oriented Software. Advances in Software Engineering, 2010. DOI:10.1155/2010/729826

[25] G. Concas, M. Marchesi, S. Pinna and N. Serra, Power-laws in a large object-oriented software system. IEEE Transaction on Software Engineering, vol.33, pp.687-708, 2007. DOI:10.1109/TSE.2007.1019

[26] G. Antoniol, M. Di Penta and E. Merlo, Predicting Refactoring Activities via Time Series. 1st International Workshop on Refactoring, 2003.

[27] B. Biegel, Q. Soetens, W. Hornig, S. Diehl and S. Demeyer, Comparison of Similarity Metrics for Refactoring Detection. Working Conference on Mining Software Repositories, 2011. DOI:10.1145/1985441.1985452

[28] T. Bodhuin, G. Canfora and L. Troiano, A tool for Suggesting Model Refactoring Actions by Metrics-led Genetic Algorithm. 1st Workshop on Refactoring Tools, pp. 23-24, 2007.

[29] M. Coraddu, F. Meloni, G. Mezzorani and R. Tonelli, Weak insensitivity to initial conditions at the edge of chaos in the logistic map. Physica A: Statistical Mechanics and its Applications 340 (1-3) , pp. 234-239, 2004. DOI:10.1016/j.physa.2004.04.012

[30] D. Dig, K. Manzoor, R. Johnson and T. Nguyen, Effective Software Merging in the Presence of Object-Oriented Refactorings. IEEE Transactions on Software Engineering, vol. 34 pp. 321-335, 2008. DOI:10.1109/TSE.2008.29

[31] A. Murgia, R. Tonelli, G. Concas, M. Marchesi and S. Counsell, An Empirical Study of Refactoring in the Context of FanIn and FanOut Coupling. 18th Working Conference on Reverse Engineering, 2011. DOI:10.1109/WCRE.2011.52

[32] A. Murgia, R. Tonelli, G. Concas, M. Marchesi and S. Counsell, Parameter-Based Refactoring and the Relationship with Fan-in/Fan-out Coupling. Fourth International Conference on Software Testing, Verification and Validation Workshops, 2011.DOI:10.1109/ICSTW.2011.26

[33] A. Murgia, R. Tonelli, G. Concas, M. Marchesi, S. Counsell, Janet McFall and Stephen Swift, Refactoring and its Relationship with Fan-in and Fan-out: An Empirical Study. 16th European Conference on Software Maintenance and Reengineering. ISBN: 978-0-7695-4666-7. DOI:10.1109/CSMR.2012.17

[34] R.E. Mullen,S.S. Gokhale, Software defect rediscoveries: a discrete lognormal model. 16th IEEE International Symposium on Digital Object Identifier, pp. 212-222, 2005. DOI: 10.1109/ISSRE.2005.37

About the authors

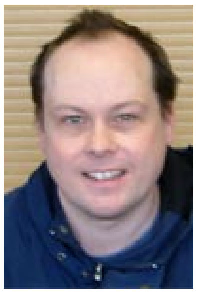
Alessandro Murgia received the laurea degree in Electronic Engineering in 2007 and his Ph.D. in Informatic Engineering in 2011. He is currently a post-doc researcher at the Electrical and Electronic Department at University of Cagliari. His research interests are refactoring, data mining from software repositories, software metrics, software bugs, and complex networks.
Contact him at `alessandro.murgia@diee.unica.it`

Roberto Tonelli received the laurea degree in Physics in 1995 and his Ph.D. in Physics in 2000, working on complex dynamical systems. He has been visitng researcher at University of Maryland, and post-doc researcher at University of California Berkeley. He is currently with the Electrical and Electronic Department at University of Cagliari - Italy, working on complex software networks and software quality.
Contact him at `roberto.tonelli@diee.unica.it`

Giulio Concas received the laurea degree in computer science from the Universita degli Studi di Pisa in 1989 with a thesis titled “Object-Oriented DataBases. From 1991 to 1999, he was the graduate technician in charge of the Scientific Calculus Area of the IT Services Center of Universita di Cagliari. Since 1995, he has performed several advanced consultancy activities in computer science field. He is a member of the Local Linux User Group and an Open Source Supporter. Since 1999, he has been assistant professor at University of Cagliari. His research interest are in Internet technology, service oriented architectures, open source, agile methodologies, and software engineering.
Contact him at `concas@diee.unica.it`

Michele Marchesi received the Laurea degrees in electronic engineering and in mathematics from the University of Genoa in 1975 and 1980, respectively. He is professor of software engineering at the University of Cagliari. His research interests include software modeling using complex system approach, agile and lean methodologies, open source development and applications, modeling and

simulation of financial markets and economic systems using heterogeneous interacting agents. He has published more than 250 papers in international journals, books and conferences. He has been the leader of several research projects amounting various million Euros, and is a consultant for various companies and public bodies. He is member of IEEE.
Contact him at michele@diee.unica.it



Steve Counsell is a Reader in the Department of Information Systems and Computing at Brunel. He received his PhD from the University of London in 2002.
His research interests are in refactoring, object-oriented metrics and applications of intelligent data analysis to software engineering problems.
Contact him at steve.counsell@brunel.ac.uk