Layer Assessment of Object-Oriented Software: A Metric Facilitating White-Box Reuse ☆

G. Kakarontzas[a,b,*], E. Constantinou[a], A. Ampatzoglou[a], I. Stamelos[a]

[a] *Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece*
[b] *Department of Computer Science and Telecommunications, T.E.I. of Larissa, 41110 Larissa, Greece*

Abstract

Software reuse has the potential to shorten delivery times, improve quality and reduce development costs. However software reuse has been proven challenging for most organizations. The challenges involve both organizational and technical issues. In this work we concentrate on the technical issues and we propose a new metric facilitating the reuse of object-oriented software based on the popular Chidamber and Kemerer suite for Object-Oriented design. We derive this new metric using linear regression on a number of OSS java projects. We compare and contrast this new metric with three other metrics proposed in the literature. The purpose of the proposed metric is to assist a software developer during the development of a software system in achieving reusability of classes considered important for future reuse and

☆This work is partially funded by the European Commission in the context of the OPEN-SME Open-Source Software Reuse Service for SMEs project, under the grant agreement no. FP7-SME-2008-2/243768.

*Corresponding author

Email addresses: gkakaron@teilar.gr (G. Kakarontzas), econst@csd.auth.gr (E. Constantinou), apamp@csd.auth.gr (A. Ampatzoglou), stamelos@csd.auth.gr (I. Stamelos)

also in providing assistance during re-architecting and componentization activities of existing systems.
Keywords: Software reuse, Object-Oriented metrics, Software metrics

1. Introduction

In Object-oriented (OO) systems objects collaborate closely in order to provide a system feature. In order to effectively reuse any class of an OO system a developer has to (a) understand the provided service of this class, (b) isolate this class from the rest of the system by extracting the class and its dependencies, (c) possibly adapt the extracted cluster of classes to the new system requirements and (d) test the class cluster to verify its correctness in the new required context. In the ideal case it would also be beneficial to transform the cluster of classes to a reusable component with provided and required interfaces, thus enabling the black-box reuse of this component in future applications. There are many difficulties associated with these activities mainly due to classes' dependencies, dependencies' dependencies and so on. These class collections can be very large and activities to understand, adapt and verify them are labor intensive. In the context of the OPEN-SME European FP7[1] project we created a unified database of metrics relating to the quality of OO software. An automated analysis tool collects the metrics and imports them in a relational database which relates metrics originating from different sources. These related metrics then, allow co-analyses using simultaneously different views of the classes. A number of projects of various sizes and domains were analyzed. The results were imported in the unified

[1] http://opensme.eu

database, and a number of recommenders were developed (with more currently under development) that access the metrics database and recommend clusters of classes that could be easily extracted from the project and transformed into reusable components. The extracted clusters are then tested, documented and placed in a code repository for future reuse by Small and Medium Enterprises (SMEs). During our work with the tools of the project we developed heuristics that could assist the component extraction activity. In most cases, classes that have a small number of dependencies and are relatively low in the system dependencies graph are easier to extract, comprehend, test etc. and consequently easier to reuse. Dependencies set cardinality and layer, unfortunately are only available *after* a project is completed. They cannot be used independently as estimators of the reusability of a class as it is constructed. We hypothesized however that there is a relationship between the Chidamber and Kemerer metrics for OO design [9] and the layer and number of dependencies. If such a relationship could be established then it would be possible to examine the reusability of a class independently during program construction. The benefit would be that a developer could be assisted in estimating the reusability of classes and if these classes were potentially useful in future applications, to be warned against problems associated to low reusability. Since architectural layers [6] are not expected to share the same reusability levels, it is also beneficial to have an indication of the expected range of reusability per layer.

In the rest of this work in Sec. 2 we provide the details of the OSS projects that we used for this study as well as details of the method for deriving the proposed facilitative metric for white-box reuse. Next in Sec. 3 we compare

our proposed metric with three other proposed reuse metrics from the literature. In Sec. 4 we examine the validity of the proposed metric separately for each of the examined projects to verify its effectiveness regardless the individual project characteristics. We also perform a calibration of the proposed metric to different project sizes and quantify the benefit that can be achieved by such a process. In the following Sec. 5 we discuss threats to validity. Then, in Sec. 6 we discuss the results and findings of this study and in Sec. 7 we discuss related work. Finally in Sec. 8 we provide future research directions and conclude.

2. Facilitating reusability assessment based on design complexity metrics

The proposed facilitative metric for white-box reuse was derived using analysis results from 29 Open Source Java projects of various sizes and application domains. The projects used, along with their sizes (number of classes excluding inner classes) and their application domains, are listed in Table 1. These projects were selected based on the following factors:

- The projects belong to different application domains since we wanted our reusability assessment to be independent as much as possible from the specifics of an application domain.
- They have different number of classes ranging from relatively small projects with tens of classes to large projects with thousands of classes.
- Most of them are mature and well-known projects, and finally
- Many of these projects were also used in other literature studies.

In total 21,775 classes were analyzed and for each class the following data were collected: (a) the Chidamber and Kemerer metrics for Object-Oriented design [9], (b) the layer of each class as reported after condensing the cyclic dependencies of the class dependency graph using the Tarjan algorithm [32], and (c) the Class Dependencies Size (CDS) metric, which is the cardinality of the dependencies set of a class, recursively following the dependencies of a class, which are necessary for a brute-force reuse of the class in a new system. Next, we briefly discuss these metrics as well as the design of the COPE tool that assisted us in collecting them. Then, we explain the rationale and the method used for deriving our proposed metric.

2.1. Metrics used

2.1.1. Chidamber and Kemerer metrics

The Chidamber and Kemerer metrics [9] were collected using the CKJM tool [30]. These metrics in general are indications of a class quality and can be used to assess reusability as well as other qualities of an OO system. The Chidamber and Kemerer suite contains six metrics which can be collected for a class during development. They are widely supported by Integrated Development Environments (IDEs) and are well accepted in the software industry. In [9] the authors characterize the effect of their metrics' suite on the reusability of classes, but without determining the importance of each metric in the class reusability estimation. More specifically, each metric along with a short description of the metric and its effect on class reusability as discussed in [9] is provided in Table 2. Along with these metrics a presumed direction of the association of each metric to reuse is provided. Later a more precise relationship will be established, which will form in fact the proposed

Table 1: OSS Projects, sizes and Domains used

No.	Project	No of Classes)	Application Domain
1	Aglets	447	Framework and environment for developing and running mobile agents
2	Ant	493	Java library and command line build tool
3	Argo UML	1578	UML modeling tool
4	Atunes	656	Audio player and organizer
5	Borg Calendar	147	Calendar and task tracking system
6	Cocoon	85	Spring-based development framework
7	Columba	1100	Email client and mail management tool
8	Compiere	2221	ERP software
9	Contelligent	836	Content Management System
10	DrJava	2207	Development environment for writing Java programs
11	EuroBudget	155	Checkbook management software
12	FreeCS	141	Chatserver (WebChat)
13	FreeMind	406	Mind-mapping software
14	GFP	312	Personal finance manager
15	JabRef	1987	Bibliography reference manager
16	JRdesktop	58	Remote desktop control, remote assistance and desktop sharing
17	jBPM	520	Business Process Management (BPM) suite
18	JEdit	508	Programmer's text editor
19	jeeObserver	172	J2EE application server performance monitoring tool
20	JMoney	54	Personal finance (accounting) manager
21	Magnolia	955	Content Management System
22	OpenEJB	1759	Enterprise Java Beans (EJB) Container System and Server
23	Projectivity	452	Enterprise Management platform
24	RapidMiner	2745	Data mining system and engine
25	Rhino	335	Implementation of Javascript written in Java
26	SportsTracker	56	Application for recording sporting activities
27	StoryTestIQ	377	Test framework to create Automated Acceptance Tests
28	SweetHome3D	167	Interior design application
29	OpenProj	846	Desktop project management application
	Total No. of Classes	21775	

facilitative metric for white-box reuse.

2.1.2. The D-Layer metric

The Classycle analyzer tool [14] is used to discover class dependencies and Directed Acyclic Graph (DAG) layers. To avoid confusion between the architecture layers of the application and the DAG layers, we call the latter D-Layers. The Classycle tool discovers strong dependencies between classes and packages, and creates a Strongly Connected Components (SCC) graph applying Tarjan's algorithm [32]. Next, according to SCCs calls, the graph is condensated to an acyclic digraph of SCCs, from which the layers are extracted (Figure 1). Although D-layers do not correspond to architectural layers since they are computed automatically from static dependencies in the source code and they do not represent actual decomposition decisions of systems' architects, they are by definition an over-approximation of the true architectural layering since they maintain one important characteristic of the architectural layering: each D-layer strictly depends upon lower D-layers and never on D-layers above it. Thus the project's layered architecture (if there is one) will be a partition of D-Layers which will, by definition of the layered architectural style, maintain the relative ordering of the D-Layers.

2.1.3. The Class Dependencies Size (CDS) metric

The last metric used in our analysis is the Class Dependencies Size (CDS). CDS is the cardinality of the dependencies set of a class. This set of classes is extracted by following the dependencies of a class recursively until we reach classes at layer 0 (i.e. classes with no remaining internal dependencies). Increased CDS is a very important indication of reuse problems. The reason is

Table 2: Metrics suite for Object-Oriented Design, [9]

Metric	Description	Effect to reusability according to [9]	Presumed direction of effect to the reusability of a class
Weighted Methods per Class (WMC)	The sum of the class's methods complexities. This results in the number of classes methods if complexity of all methods is considered to be equal to 1.	According to [9] "Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse."	Negative
Depth of Inheritance Tree (DIT)	The depth of inheritance of the class in the inheritance hierarchy. Notice that in the case of the Java programming language this is at least equal to 1 since every class implicitly inherits from java.lang.Object. This is also true for other OO languages such as C#.	The authors do not explicitly state the effect of DIT to reuse. They only mention reuse in relation to inherited methods: "The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods" [9]	Inconclusive: Indeed DIT can be viewed from different perspectives. Consider a framework which is specifically designed for reuse. In a framework a class may have large DIT values yet it is heavily reused. There are cases however where large DIT values signify application specific classes.
Number of Children (NOC)	The number of the immediate subclasses of the class.	"Greater the number of children, greater the reuse, since inheritance is a form of reuse." [9]	Positive
Coupling Between Objects (CBO)	The count of the number of other classes a class is coupled to.	"Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application." [9]	Negative
Response set For a Class (RFC)	The number of methods that can potentially be executed in response to a message received by the class. Notice that this involves only the methods of a class and the methods that these methods potentially call due to practical difficulties in computing the metric. Also notice that the CKJM tool that we used to collect this metric also follows this approach.	The authors mention that higher RFC values imply increased effort for testing and debugging. Although this suggests a possible relation to white-box reuse, the authors do not explicitly mention reuse.	Negative: Although the authors do not explicitly mention reuse, the effort related to RFC for increased testing and debugging can be important in the context of white-box reuse in which the reuser needs to adapt the reused classes to a new slightly different context. Therefore we assume that RFC is negatively related to reuse.
Lack of Cohesion of Methods (LCOM)	The count of the number of method pairs in a class whose similarity is 0 minus the count of method pairs whose similarity is not 0. Similarity of methods is established by means of common instance variables accesses.	The authors do not mention reuse explicitly.	Inconclusive: The Lack of Cohesion of Methods can be considered a negative characteristic to reuse (e.g. as in controllers where a controller delegates messages to a number of unrelated domain classes). But reuse is not mentioned explicitly in [9].

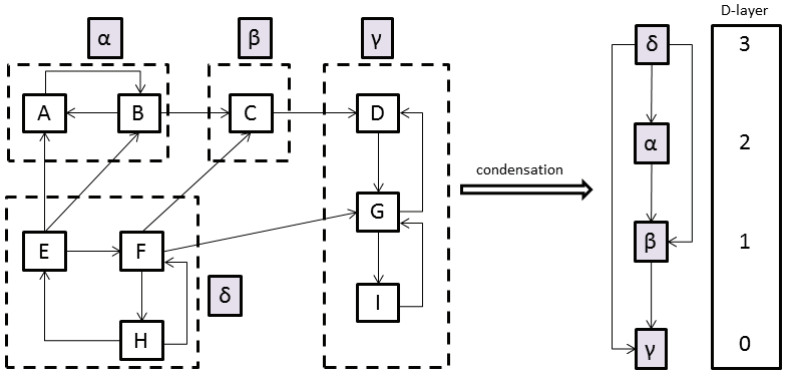

Figure 1: Classycle Acyclic Digraph Creation [14]

that class reuse involves comprehending, testing, adapting etc. the whole dependencies set, unless the class in question is already a black box component or an extension point for a framework already designed for reuse. In these cases CDS is irrelevant for external reusers that simply follow the documentation to reuse the component or framework. However it is still relevant for the developers of the component or the framework, who need to maintain and evolve the component or framework code internally.

It is important at this point to make the following distinction: our work is not relevant for already formed black-box components and frameworks and therefore cannot be used to assess the reusability of these artifacts from the standpoint of external reusers. However, it can still be relevant for the purposes of maintaining or reusing parts of these artifacts. Current work is mostly suitable for facilitating the assessment of the reusability of classes possibly appropriate for becoming extension points of frameworks or facades of black-box components. Thus it provides a tool in the context of system improvement by re-architecting and componentization activities. It is also

benefitial for developers, as they develop classes iteratively during a project lifecycle when the system is not fully formed. Developers are assisted in estimating the reusability of classes that may be important from a reusability perspective.

2.2. The COPE tool

The aforementioned metrics are combined in a relational database and are analyzed by a tool called the Component Adaptation Environment (COPE). COPE also integrates various tools, such as the CKJM tool [30] and Classycle [14], to provide a more user friendly and productive environment. The relational database of COPE is depicted in Figure 2. As can be seen there are projects which are related to packages and classes. Packages have internal dependencies to other project packages and classes to other project classes. A class is also related to its package. We also maintain the project history in the version control system (the logs, logentries and paths tables), and relate each path to the class file that was affected by the respective log entry. However, in this work recommenders related to the project's history are not discussed. Classes table not only retains the dependencies of each class and the D-Layer, but also stores all the class CK metrics. COPE tool therefore, consolidates in its database information from different sources (e.g. Classycle, CKJM, Version Control System etc.) and allows different types of recommenders to be constructed that recommend clusters for component extraction based on the consolidated metrics.

COPE provides a graphical user interface (Figure 3) from which the reuse engineer can import a project and analyze it, get recommendations and extract components. Components can also be tested using advance Model-

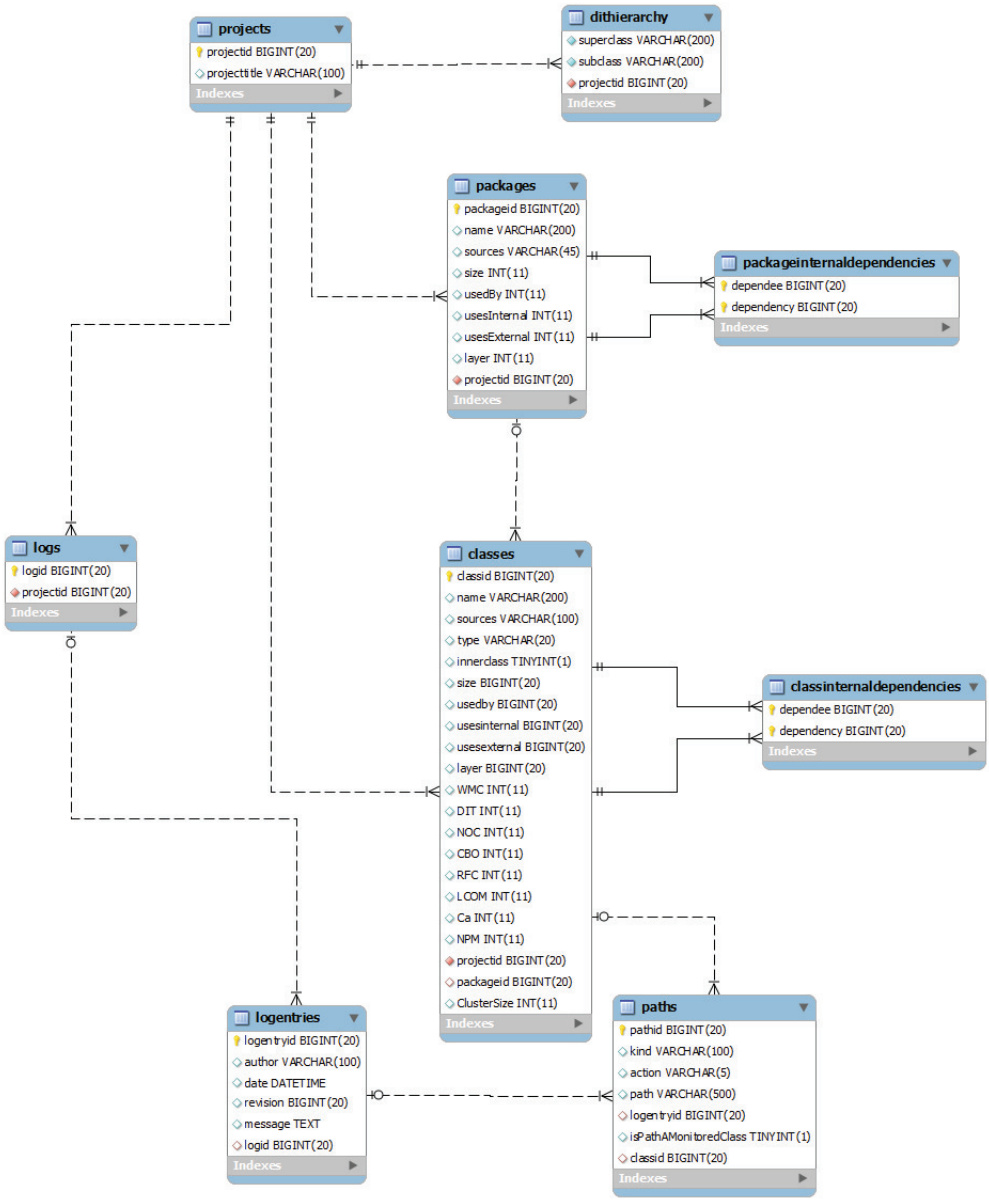

Figure 2: COPE database

Based Testing techniques [7]. The extracted components and their documentation are then packaged and imported in a repository for future reuse.

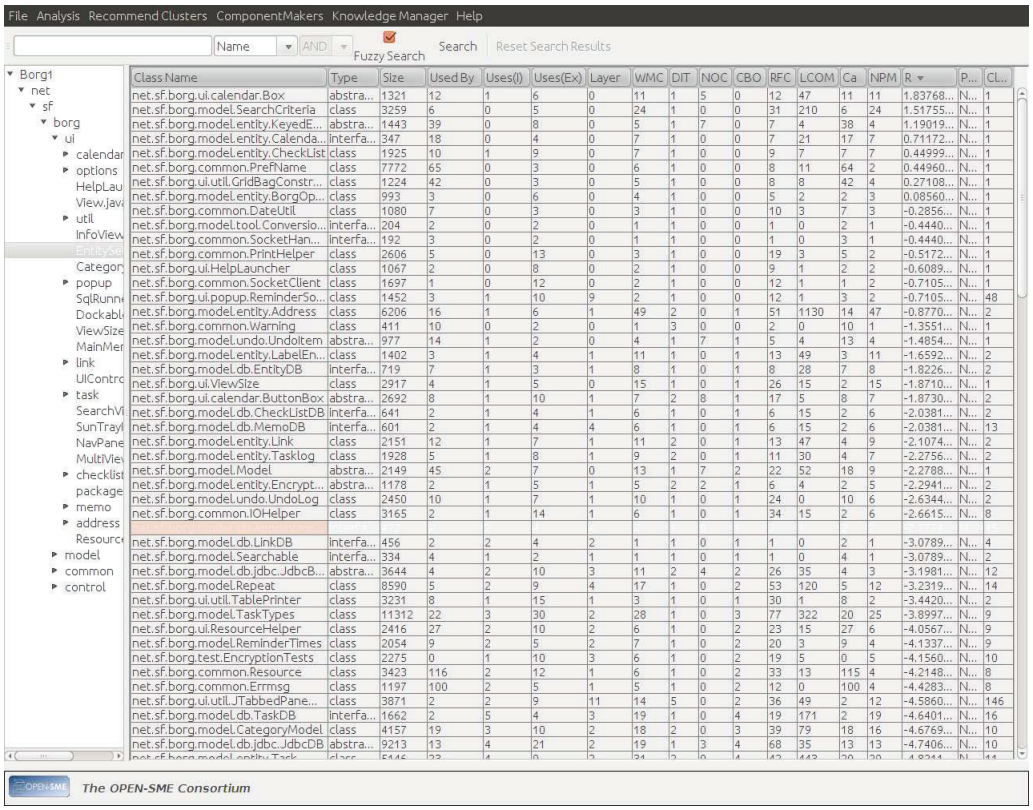

Figure 3: The COPE tool

2.3. The proposed facilitative metric for white-box reuse

In this work we make the assumption that a lower D-layer is a necessary (but not sufficient) factor to improve reusability. This is something that is peripherally discussed or implied in a number of works. For example Selby in [29] reports that "The module design factors that characterize module reuse without revision were (after normalization by size in source lines): few

calls to other system modules, many calls to utility functions, few input-output parameters, few reads and writes, and many comments". Some of these characteristics, especially the few calls to other system modules and the many calls to utility functions, indicate modules closer to the bottom of a layered architecture. It is also well known that lower layers in systems following the layered architectural style have fewer dependencies and thus are more reusable [26, 22]. Although it is well known and expected that classes with smaller D-Layer will generally have fewer dependencies we investigated the extent of the statistical correlation of the D-Layer with the CDS value of each class. We used non-parametric Spearman ρ and the details per project are depticted in Table 3.

Table 3 shows that the correlation is very strong and significant (0.01) in all projects. Therefore the negative relationship between the class D-layer and the reusability of the class is established by the fact that higher D-layers imply larger dependencies sets and therefore increased effort for comprehension, testing, debugging and adaptation of the reused class to a new system. In other words these results verify what is known from practice: classes at lower layers are usually easier to reuse due to their fewer dependencies.

To derive the proposed facilitative metric for reuse a linear regression analysis was performed using the Chidamber and Kemerer (CK) metrics collected from the above mentioned projects as predictors of the D-Layer variable. According to [34], one of the first steps during statistical analysis of the dataset is the elimination of outliers. This procedure is part of data reduction phase [34], that is crucial in order for the results of the empirical study not to be affected by characteristics of individual cases. Therefore, to

Table 3: D-Layer and CDS correlation (Spearman ρ - $p < 0.01$)

Project No.	D-Layer-CDS correlation (Spearman ρ)	Project No.	D-Layer-CDS correlation (Spearman ρ)
1	0.99	16	0.978
2	0.97	17	0.974
3	0.992	18	0.994
4	1.0	19	0.995
5	0.995	20	0.970
6	0.980	21	0.950
7	0.983	22	0.979
8	0.988	23	0.963
9	0.942	24	0.982
10	0.889	25	0.803
11	0.944	26	0.981
12	1.0	27	0.998
13	0.999	28	0.994
14	0.980	29	1.0
15	0.806		

avoid possible problems caused by outliers of the CK values we first used an analysis that removed the outliers (i.e. classes that had unusually large or small values for the CK metrics). In order to identify the outliers for each metric we have created the corresponding boxplots and identified the boundary metrics scores that are not characterized as outliers. The filter that has been used as inclusion criterion is presented in the logical expression in Eq. 1. The observed extreme values where identified as maximum values, that is classes exhibiting unusually large metric scores.

$$\begin{aligned}(WMC < 16) \wedge (NOC < 4) \wedge (DIT < 7) \wedge \\ (CBO < 14) \wedge (RFC < 60) \wedge (LCOM < 37)\end{aligned} \tag{1}$$

In Figures 4 and 5 we present the boxplots for the six CK metrics before and after the elimination of the outliers.

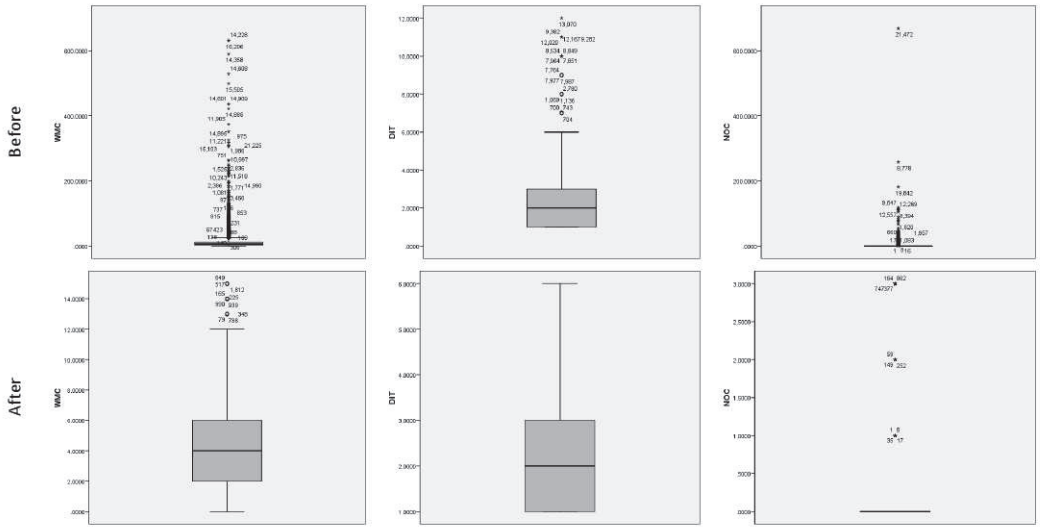

Figure 4: Boxplots of WMC, DIT and NOC before and after outlier elimination

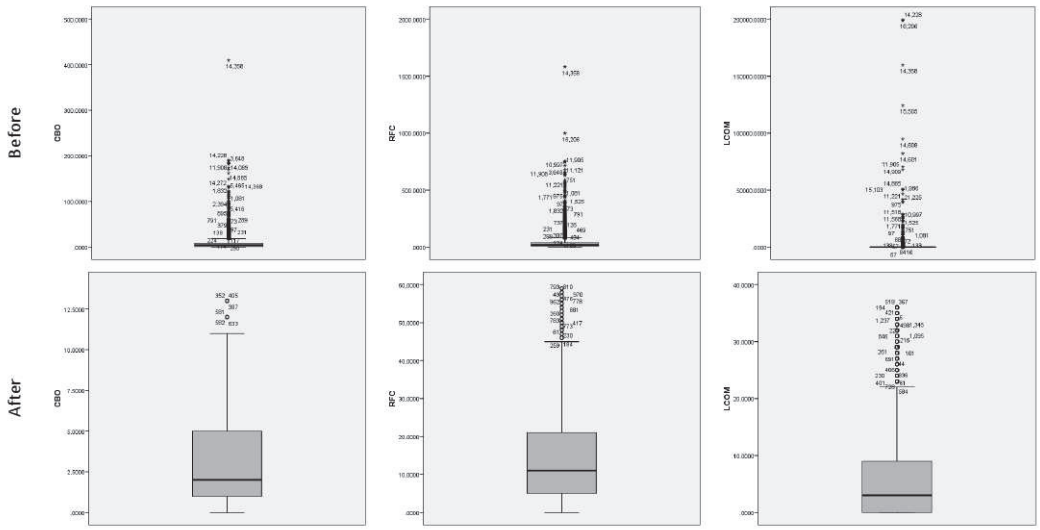

Figure 5: Boxplots of CBO, RFC and LCOM before and after outlier elimination

The filtering of the outliers excluded 7,636 classes. The remaining 14,139 classes were used for the regression analysis. It is important to note here that the removed classes do not represent 'wrong' or 'misunderstood' observations. In such cases the authors of [34] suggest that they should be excluded. Although the outliers were legitimate cases they were nevertheless extreme and their presence would have affected the metric. Since we wanted our metric to be valid for the 'typical' case we have decided to remove the outliers from our study. After the exclusion of the 7,636 cases, the number of the remaining outliers did not significantly affect our study. Since the removed outliers represent in some sense unusual yet legitimate cases, it would be interesting in a follow up study to examine only the removed classes in relation to their specific characteristics and the reasons that they have these unusual complexity metrics. Forward regression was applied on the logarithmic transformations

of the CK metrics as predictors of the D-Layer dependent variable. We have used a logarithmic transformation of the predictor variables in order to correct problems with the normality of data and lessen the effect of any remaining outliers in the data [15]. The forward linear regression analysis was performed through the origin (no intercept). This analysis resulted in entering all six predictors (i.e. $log(CBO+1)$, $log(DIT+1)$, $log(WMC+1)$, $log(RFC+1)$, $log(LCOM+1)$, and $log(NOC+1)$), with a final $R^2 = .714$. The coefficients of the predictors along with the t-statistic, the standard error and the significance are reported in Table 4. The proposed Facilitative metric for White-Box Reuse (FWBR) is therefore formulated as shown in Eq. 2.

Table 4: Regression analysis results

Predictor	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	Std. Error	Beta		
$log(CBO+1)$	8.753	.122	.764	71.735	.000
$log(DIT+1)$	2.505	.158	.177	15.830	.000
$log(WMC+1)$	-1.922	.230	-.198	-8.338	.000
$log(RFC+1)$	.892	.167	.140	5.332	.000
$log(LCOM+1)$	-.399	.081	-.042	-4.912	.000
$log(NOC+1)$	-1.080	.312	-.016	-3.463	.001

$$
\begin{aligned}
FWBR = -1 \times (&8.753 \times \log(CBO + 1) \\
&+ 2.505 \times \log(DIT + 1) \\
&- 1.922 \times \log(WMC + 1) \\
&+ 0.892 \times \log(RFC + 1) \\
&- 0.399 \times \log(LCOM + 1) \\
&- 1.080 \times \log(NOC + 1))
\end{aligned}
\tag{2}
$$

In Eq. 2, the D-Layer or each class is predicted, using the the CK metrics as estimators. Notice that the regression analysis was carried out using the D-Layer as the predicted variable and D-Layer is strongly correlated to the Cluster Dependencies Size (CDS) as can be seen from Table 3. Since CDS in negatively related to reuse, the direction of the formula is reversed by multiplying by -1 to obtain the metric. The idea is simply that we consider classes more reusable, if reusing them requires fewer dependencies to understand, adapt, test etc. Therefore the proposed metric favors classes, with fewer total dependencies. However the proposed metric does not require obtaining CDS, which is labor intensive and requires the whole project to be available.

The regression analysis in our sample provides evidence for a quantitative assessment of the effect of each CK metric to the reusability of OO software. Returning in our hypotheses from Table 2 and based on the sample of the $21,775$ classes analyzed, we can say the following:

- *CBO*: Coupling Between Objects is the most negatively influential factor for the reuse of classes in our sample. A high CBO value indicates

classes that reusing them requires understanding, testing, adapting etc. large numbers of additional classes. This conclusion is also similar to conclusions suggested by other related works such as [29] and [18].

- *DIT*: Depth of Inheritance Tree is the second most significant factor that prevents reuse in our sample. Although as we mentioned in Table 2 DIT can be viewed from different perspectives, it seems that when it comes to white box reuse it is a very important negative factor. Gray box reuse (e.g. frameworks) and black box reuse (e.g. components) on the other hand are not affected by DIT since the internal details of the reused source code are irrelevant.

- *WMC*: Weighted Methods per Class is the third most influential factor in reuse in our sample, but this time in a positive direction. This conclusion is rather surprising since it seems that the larger the number of methods of a class the more reusable it is. However it is consistent with the view adopted in other works that a large number of public methods signifies a more reusable class. For example [3] links reusability in a positive way with Class Interface Size which is the count of the number of public methods in a class. Similarly [2] discusses how WMC can be viewed from different perspectives. On the one hand it can be viewed as a complexity factor, but on the other hand it can be said that classes with higher WMC are more rewarding for reuse since they provide more services to the reusers. Our findings verify that WMC is in fact positively related to white-box reuse. A slightly different interpretation is that the lower a class is in the DAG of the project the

more the pressure to provide additional services to classes above it will be, since there are more opportunities to refer to it.

- *NOC*: Number of Children signifies that a class is internally reused. It is the fourth more influential factor to reuse according to our sample analysis in a positive way. Indeed this is consistent with [9].

- *RFC*: Response set for a Class has the expected direction in our sample: it is slightly negatively related to reuse. As we mentioned in Table 2 we expected that RFC would have a negative relation to reuse, since we are concerned with white-box reuse and [9] mentions that higher RFC values imply increased effort for testing and debugging.

- *LCOM*: Lack of Cohesion of Methods coefficient is very small, only -0.399 and is in our result the least influential factor. Although its direction is (according to the findings) positive, it accounts only for 2% of the influence and therefore it is rather insignificant. However it certainly seems that LCOM at least as suggested by [9] is not negatively related to white box reuse which seems to support [9] that did not mention reuse explicitly in relation to LCOM and contradicts other works such as [3] who, perhaps reasonably, assumed that cohesion is positively related to reuse.

Table 5 provides the descriptive statistics of the projects used: all the CK metrics along with the descriptive statistics for the new proposed metric $FWBR$.

In our analyses we first thought that we should use CDS directly as the predicted variable in regression analysis. However in almost all the projects

Table 5: Descriptive statistics for the CK and the new reuse metrics ($N = 21,775$ classes)

	Range	Min	Max	Mean	Std. Deviation	Variance
WMC	632	0	632	10.395867	18.5326263	343.458
DIT	11	1	12	2.467279	1.7783182	3.162
NOC	668	0	668	.511871	5.8086298	33.740
CBO	410	0	410	6.540758	10.1383096	102.785
RFC	1,581	0	1,581	30.909759	44.2458528	1,957.695
LCOM	199,396	0	199,396	174.376211	2,819.6823013	7,950,608.280
FWBR	23.13	-20.59	2.54	-6.0640	3.81117	14.525

analyzed we observed a phenomenon that would eliminate most of the higher layer classes from our regression analysis. The problem is that in most projects above a certain layer, CDS was more or less the same for all classes, making these classes indistinguishable for analysis purposes. A possible explanation for this phenomenon is that certain classes are hubs through which all the calls to lower layers are delegated. Therefore including such a class in a cluster attracts the rest of the system with it, and therefore CDS remains more or less the same after a certain layer. To give an example of this, in the Argo UML project 761 classes starting from layer 10 and above have $803 \leq CDS \leq 810$. Classes below D-layer 10 have a CDS value under 60. These 761 classes represent 47% of the project classes in our analysis. This similarity therefore would have had a significant impact in the regression analysis results. We preferred therefore obtaining the regression analysis results based on the D-Layer which, as shown in Table 3, has a very strong

and significant correlation with CDS. The CDS relationship with D-Layer is depicted in Fig. 6, in which D-Layer is in the x-axis and the median of CDS in the y-axis.

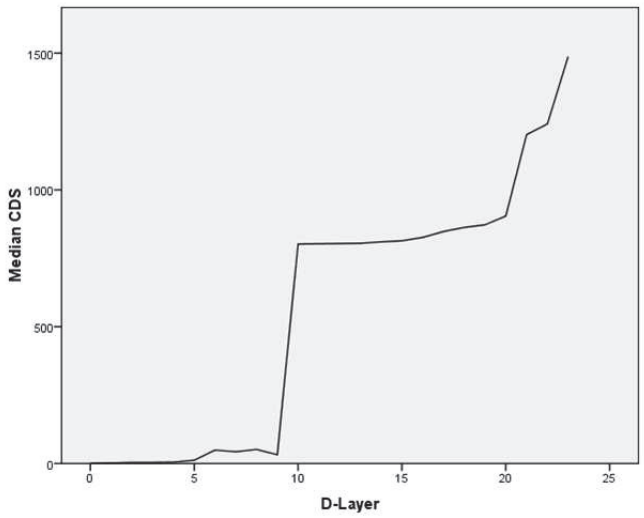

Figure 6: CDS Median per D-Layer

From Fig. 6, we can see that a large number of classes in higher layers are difficult to reuse since reusing them requires a large part of the system to be present. These classes are therefore application specific, which is also another indication that a white-box reuse metric should be strongly negatively correlated to CDS.

In the next Section the proposed metric is compared to three other metrics proposed in the literature.

3. Comparative assessment of the proposed metric

In order to assess the effectiveness of the proposed FWBR metric it was compared to three other metrics proposed in the literature. The comparison

involved the following:

- *Test 1*:How well each of the proposed metrics correlates to the CDS for each class? A good reuse metric should have a significant negative correlation to CDS because it should favor classes that generally have fewer requirements.

- *Test 2*: How well each of the proposed reuse metrics correlates to the D-Layer and arbitrary partitions of the D-Layer? A good reuse metric should have a significant negative correlation to the architectural layers of the application. Since D-Layer represents an over-approximation of the layered architecture it follows that it should also have negative correlation to the D-Layer and arbitrary partitions of the D-Layer. For the comparisons, the original D-Layer of each project as well as partitions of equal size of the D-Layer structure in 5, 6, 7 and 8 equal chunks were used.

- *Test 3*: Given the D-Layer how well each of the proposed reuse metrics correlates to CDS? Naturally it is expected that the most reusable classes at each layer should have fewer requirements in relation to the less reusable classes. In other words it is expected that at each individual layer the reuse metric should be negatively correlated to CDS.

The metrics used for the comparison are the following:

- *Bansiya [3]*: In this work the authors propose a hierarchical quality model for Object-Oriented software. In such a model, every high level quality attribute, such as reusability, is calculated as a function of low

level quality attributes. The authors suggest that reusability represents the existence or absence of object-oriented characteristics that allow a design to be reapplied to a new project without a significant effort. The reusability as defined and calculated in QMOOD, takes into account structural quality characteristics such as coupling and cohesion that are very important when applying white-box reuse. According to QMOOD software reusability is calculated as shown in Eq. 3.

$$\begin{aligned} Reusability = 0.25 \times CAMC + 0.5 \times CIS \\ + 0.5 \times DSC - 0.25 \times DCC \end{aligned} \tag{3}$$

In Eq. 3:

  - CAMC (Cohesion among Methods of the class) Is calculated as a fraction. The numerator is the sum of distinct parameter types in all methods of a class. The denominator is calculated by multiplying the number of methods by the number of distinct type of parameters in the class.
  - CIS (Class Interface Size): Is the count of public methods of a class
  - DSC (Design Size in Classes): Is the count of classes in the design
  - DCC (Design Class Coupling): Is the count of other classes that one class is directly related to.

- *Inoue [19]*: In this work the authors are concerned with ranking the significance of software components that are the results of a search query. Their idea was that the most reusable components returned

should be ranked higher and in order to achieve that they devised an algorithm for computing reusability based on use relations. The authors model a software system as a component graph G = (V, E) that is weakly connected, where each class is considered a component. Components form the graph nodes and edges are defined according to their use relations. If between two nodes a use relation does not exist, the authors consider pseudo use relations.

Initially, each node weight $w'(v_i)$ is defined as $w'(v_i) = \frac{1}{n}$, where n is the number of system classes.

The edge weights are calculated according to the use relation between the nodes. If there is no use relation between nodes v_i and v_j then $w'(e_{ij})$ is defined as in Eq. 4.

$$w'(e_{ij}) = \frac{1}{n} \tag{4}$$

If a use relation exists from node v_i to node v_j, the edge weight $w'(e_{ij})$ is defined as shown in Eq. 5.

$$\begin{aligned} w'(e_{ij}) &= d'_{ij} \times w'(v_i), \\ d'_{ij} &= p \times d_{ij} + \frac{1-p}{n}, \\ d_{ij} &= \frac{w(v_i)}{OUT(v_i)} \end{aligned} \tag{5}$$

In Eq. 5 p, $(0 < p < 1)$ is the ratio of real use relations and pseudouse relations, and the authors employ a fairly large value $p = 0.85$. The authors examined the fact that the resulting weight nodes are affected by p, but the final component ranks are insensitive to p. They have chosen $p = 0.85$, since the ranks are fairly stable for p values from 0.75

to 0.95. $OUT(v_i)$ in Eq. 5, is the number of elements in the set of the outgoing edges from node v_i.

The following iterative procedure is performed until next-step node weights are the same as the previous ones: Each node weight is redefined as the sum of weights of incoming edges according to Eq. 6 and the edges weights are recomputed according to Eq. 4 and Eq. 5.

$$w'(v_i) = \sum_{e_{ki} \in IN(v_i)} w'(e_{ki}) \tag{6}$$

In Eq. 6, $IN(v_i)$ is the set of incoming edges to node v_i.

The resulting metric is based on statical use relations and the weights calculated depend on the fan-in of a class and the weights of these classes.

- *Nair [27]*: This work proposes a reusability index based on the the empirical investigation of two industrial projects. The first project consists of 265 classes of which 96 classes are reusable. The second consists of 423 classes of which 198 are reusable. The analysis included expert opinions on the reusability of each class, however details of the projects were not provided due to a Non-Disclosure Agreement. The analysis resulted in Eq. 7 in which the reusability index of a class (CRul) is computed from three metrics of the Chidamber-Kemerer suite, namely

DIT, RFC and WMC.

$$
\begin{aligned}
RuDIT &= 7.8 \times DIT^{1.4}, \\
RuRFC &= 0.006 \times RFC^2 \\
&\quad + 1.23 \times RFC + 11.1, \\
RuWMC &= 0.001 \times WMC^3 \\
&\quad - 0.16 \times WMC^2 + 7 \times WMC - 2.7, \\
CRul &= 0.33 \times RuDIT + 0.27 \times RuRFC \\
&\quad + 0.4 \times RuWMC
\end{aligned}
\tag{7}
$$

Two notable differences from the proposed metric in this work are that CBO is not included when in our proposed metric is the most influential factor, and that in Nair metric, DIT is associated positively to the reusability of a class.

Although the aforementioned metrics are comparable to the proposed metric since they use comparable code or design characteristics and assess reusability, we should mention that the work by Inoue et al. [19] starts with a different goal in mind: in [19] the authors aim at supporting a search engine and propose a component rank as an indication for the positioning of the search result (high or low) in the results list. They are not directly concerned with the reusability assessment of the source code for extensions or modifications. However their assessment is also based in class relations and they mention that the proposed "component rank can be considered as a true measure of software reuse". We therefore considered it comparable to our proposed metric.

In the following subsections we present the correlations for the tests mentioned earlier using the Spearman non-parametric test (one-tailed).

3.1. Test 1

The first comparison examines if the proposed metric outperforms other proposed metrics, in relation to the Cluster Dependencies Size (CDS) of a class. As can be seen in Table 6 the proposed metric outperforms the three other proposed metrics in relation to CDS. In this analysis all $21,775$ classes were considered and $FWBR$ is significantly negatively correlated ($\rho = -.657, p < 0.01, 1 - tailed$) to CDS. Bansiya's metric [3] is also negatively correlated but with a weaker correlation ($\rho = -.226, p < 0.01, 1 - tailed$) and the same holds for Inoue [19] ($\rho = -.180, p < 0.01, 1 - tailed$). On the other hand the reuse metric proposed in [27] is positively correlated to CDS according to our analysis ($\rho = .359, p < 0.01, 1 - tailed$).

This analysis verifies that using the proposed metric will indicate classes that are more probable to attract fewer classes in total when reused in another context.

Table 6: Spearman's ρ for the reuse metrics correlation to CDS

Spearman's ρ					
		Nair	**Inoue**	**Bansiya**	**FWBR**
CDS	Correlation Coefficient	.359**	-.180**	-.226**	-.657**
	Sig. (1-tailed)	.000	.000	.000	.000
	N	21775	21775	21775	21775

****. Correlation is significant at the 0.01 level (1-tailed).**

3.2. Test 2

As a second comparison of the metrics, a correlation analysis was performed with the D-Layer of each class and different partitions of the D-Layer that maintain the relative order of the dependencies. These partitions are an attempt to simulate the architectural layers of an application which are significantly fewer than the D-Layers. The D-Layer structure was partitioned to different sizes (i.e. 5, 6, 7 and 8 parts) to simulate different candidate layered architectures. It is important to note that none of these partitions actually corresponds exactly to a true layered architecture since they do not represent the actual architectural partitions of the systems' architects, but they partition the classes to larger groups of classes than the D-Layers which maintain the essential property of the layered architecture (i.e. classes at each layer refer or use classes only at lower layers and never on layers above them).

The results are again very favorable for the proposed $FWBR$ metric which shows a strong negative correlation to all layered partitions and the original D-Layer. Again the Bansiya and Inoue metrics are outperformed by $FWBR$ but they are also significantly negatively correlated to the different partitions albeit with a weaker correlation compared to $FWBR$. Nair metric exhibits a positive correlation to the layered partitions. The details of this analysis can be seen in Table 7.

The negative correlation with the different layered partitions are favorable to reuse since it is well known that classes at lower layers are more reusable than classes at higher layers of the system layered architecture [6, 29, 26, 22].

Table 7: Spearman's ρ for the reuse metrics correlation to different layer partitions

Spearman's ρ					
		Nair	**Inoue**	**Bansiya**	**FWBR**
D-Layer	Correlation Coefficient	.349**	-.278**	-.235**	-.654**
	Sig. (1-tailed)	.000	.000	.000	.000
	N	21775	21775	21775	21775
DLayer5	Correlation Coefficient	.306**	-.250**	-.311**	-.644**
	Sig. (1-tailed)	.000	.000	.000	.000
	N	21717	21717	21717	21717
DLayer6	Correlation Coefficient	.310**	-.239**	-.286**	-.641**
	Sig. (1-tailed)	.000	.000	.000	.000
	N	21491	21491	21491	21491
DLayer7	Correlation Coefficient	.322**	-.244**	-.300**	-.660**
	Sig. (1-tailed)	.000	.000	.000	.000
	N	21491	21491	21491	21491
DLayer8	Correlation Coefficient	.332**	-.274**	-.281**	-.668**
	Sig. (1-tailed)	.000	.000	.000	.000
	N	20835	20835	20835	20835

****. Correlation is significant at the 0.01 level (1-tailed).**

3.3. Test 3

In the third test the data was split to D-Layers and for each individual D-Layer a correlation analysis was performed to CDS per project. This comparison attempts to capture the more reusable classes per layer. Since classes at the same D-layer of a project are classes that have the same role in the same system they are more similar (e.g. GUI classes, controllers, entities etc.) and a successful reuse metric should highlight the more reusable among them favoring those with fewer dependencies. Since current work is interested in white-box reuse as a starting point for componentization, re-architecting and framework extraction, we want each class to require fewer dependencies to be reused. Notice that for at least two reasons the correlation of a reuse metric to CDS may not be easy to establish separately for each D-Layer:

1. First the classes of a D-Layer may be very few (e.g. 8 classes) in which case a significant correlation cannot be established.
2. Second, as discussed earlier, in higher D-Layers the dependencies of each class belonging to this D-Layer may be the same or very close to the dependencies of any other class belonging to the same D-Layer. This may be the result of a design that overlooked dependencies control and is something that we noticed in the majority of the analyzed projects. Again the correlation cannot be established since CDS does not differ significantly for the classes of the analyzed D-Layer.

For these reasons the failure to establish a significant negative correlation between a reuse metric and CDS is not necessarily indicative of problems with the metric. However if a metric correlates positively to CDS then this may be an indication of problems in using the metric for white-box reuse.

The reason is that the metric will highlight classes as reusable when these classes require more dependencies to be reused.

First, to avoid the second problem, a correlation analysis was performed in two projects that included only the lower 7 D-layers. The first project was a medium sized project (Aglets) with 447 classes. The second project was a larger project (JabRef) with 1,978 classes. The results are depicted in Tables 8 and 9.

In the case of the Aglets project (Table 8) the proposed metric was correlated significantly with a negative correlation to CDS in 3 of the 7 D-layers at 0.01 level and in 1 D-layer with a significant negative correlation at 0.05 level. None of the compared metrics established a significant negative correlation. On the contrary Nair was correlated significantly at the 0.01 level with a positive correlation to CDS in 4 D-Layers, Inoue in 2 D-Layers and Bansiya in 1 D-Layer. The proposed metric had no significant positive correlations to CDS.

In the case of the JabRef project (Table 9) the proposed metric was correlated negatively to CDS with significance at the 0.01 level 6 out of the 7 D-Layers and in one case (D-Layer 6) negatively with significance at the 0.05 level. Bansiya was correlated negatively to CDS with significance at the 0.01 level 3 times and with significance at the 0.05 level one time. Inoue was correlated negatively to CDS with significance at the 0.01 level 2 times and Nair 1 time. Again however, Inoue and Nair exhibited positive correlations to CDS in 2 cases each at the 0.01 level. The proposed metric again was never correlated positively to CDS.

Table 8: Aglets project: CDS Spearman's ρ

Project Name	**Aglets**					
Size	**447 classes**					
			Nair	**Inoue**	**Bansiya**	**FWBR**
Layer 6	**CDS**	Correlation Coefficient	0.599**	0.427*	0.272	-0.295
		Sig. (1-tailed)	0.001	0.015	0.09	0.072
		Number of classes	26	26	26	26
Layer 5	**CDS**	Correlation Coefficient	0.641**	0.426**	-0.11	-0.593**
		Sig. (1-tailed)	0	0.001	0.212	0
		Number of classes	55	55	55	55
Layer 4	**CDS**	Correlation Coefficient	-0.002	0.452**	-0.031	-0.14
		Sig. (1-tailed)	0.494	0	0.405	0.135
		Number of classes	64	64	64	64
Layer 3	**CDS**	Correlation Coefficient	0.555**	0.166	0.564**	-0.611**
		Sig. (1-tailed)	0.005	0.237	0.004	0.002
		Number of classes	21	21	21	21
Layer 2	**CDS**	Correlation Coefficient	0.196	0.313*	0.335*	-0.516**
		Sig. (1-tailed)	0.137	0.038	0.028	0.001
		Number of classes	33	33	33	33
Layer 1	**CDS**	Correlation Coefficient	0.318**	0.143	0.338**	-0.069
		Sig. (1-tailed)	0.005	0.13	0.003	0.293
		Number of classes	64	64	64	64
Layer 0	**CDS**	Correlation Coefficient	0.008	0.234*	0.037	-0.214*
		Sig. (1-tailed)	0.47	0.017	0.371	0.027
		Number of classes	82	82	82	82

****. Correlation is significant at the 0.01 level (1-tailed).**

***. Correlation is significant at the 0.05 level (1-tailed).**

Table 9: JabRef project: CDS Spearman's ρ

Project Name	**JabRef**					
Size	**1,978**					
			Nair	**Inoue**	**Bansiya**	**FWBR**
Layer 6	**CDS**	Correlation Coefficient	0.048	0.5**	-0.007	-0.198*
		Sig. (1-tailed)	0.345	0	0.476	0.046
		Number of classes	73	73	73	73
Layer 5	**CDS**	Correlation Coefficient	0.334**	-0.16	-0.325**	-0.38**
		Sig. (1-tailed)	0.002	0.089	0.003	0.001
		Number of classes	72	72	72	72
Layer 4	**CDS**	Correlation Coefficient	-0.046	-0.31**	-0.454**	-0.531**
		Sig. (1-tailed)	0.298	0	0	0
		Number of classes	136	136	136	136
Layer 3	**CDS**	Correlation Coefficient	-0.183**	-0.54**	-0.441**	-0.553**
		Sig. (1-tailed)	0.002	0	0	0
		Number of classes	248	248	248	248
Layer 2	**CDS**	Correlation Coefficient	0.111*	-0.06	0.128*	-0.581**
		Sig. (1-tailed)	0.027	0.161	0.013	0
		Number of classes	301	301	301	301
Layer 1	**CDS**	Correlation Coefficient	0.196**	0.109	0.145*	-0.296**
		Sig. (1-tailed)	0.005	0.077	0.028	0
		Number of classes	174	174	174	174
Layer 0	**CDS**	Correlation Coefficient	0.085	0.36**	0.115**	-0.387**
		Sig. (1-tailed)	0.076	0	0.025	0
		Number of classes	289	289	289	289

****. Correlation is significant at the 0.01 level (1-tailed).**

***. Correlation is significant at the 0.05 level (1-tailed).**

Next a correlation analysis was performed to the first 10 projects of Table 1, and the following were measured:

1. The percentage of negative correlations significant at the 0.05 level.
2. The percentage of negative correlations significant at the 0.01 level.
3. The percentage of negative correlations significant at the 0.01 level that were at the same time stronger than 0.4.

The aforementioned negative correlations are considered favorable for a metric facilitating white-box reuse since the metric value should be smaller for classes with larger required dependencies (i.e. increased CDS). As we mentioned already it wasn't expected to find strong and significant correlations for all the D-layers analyzed for the two aforementioned reasons. However, it was important to analyze if the reuse metrics were highlighting incorrectly classes with larger dependencies (i.e. the correlation of the reuse metric to CDS was positive). Therefore the following were also measured:

1. The percentage of positive correlations significant at the 0.05 level.
2. The percentage of positive correlations significant at the 0.01 level.

Table 10 and Fig. 7, show the count and percentages of the negative correlation per metric for the 164 D-layers that were analyzed. It can be seen that the proposed metric exhibits significantly stronger negative correlations to CDS per layer and per project. From the 164 D-layers examined, the $FWBR$ metric was significantly negatively correlated at the 0.01 level 42 times (25.61%), whereas the compared metrics were significantly negatively correlated in fewer than 6% of the cases. The proposed metric also outperforms the compared metrics in significant negative correlations at the 0.05

level and, significant negative correlations at the 0.01 level with strength more than 0.4.

Table 10: Percentages of negative correlations per layer and per project to the reuse metrics

	Negative correlations significant at 0.05 level	**Negative correlations significant at 0.05 level (%)**	**Negative correlations significant at 0.01 level**	**Negative correlations significant at 0.01 level (%)**	**Negative Correlations significant at 0.01 level stronger than .4**	**Negative Correlations significant at 0.01 level stronger than .4 (%)**
Nair	4	2.44%	1	0.61%	0	0.00%
Inoue	4	2.44%	4	2.44%	1	0.61%
Bansiya	5	3.05%	9	5.49%	6	3.66%
FWBR	19	11.59%	42	25.61%	28	17.07%
					Total Layers examined	164

Table 11 and Fig. 8, on the other hand show the positive (i.e. false) correlations that were observed. As can be seen the proposed reuse metric was correlated positively with significance at the 0.05 level only at 4 D-Layers from the total 164 D-Layers examined, and never with a significance at the 0.01 level. The performance of the compared metrics was less than optimal

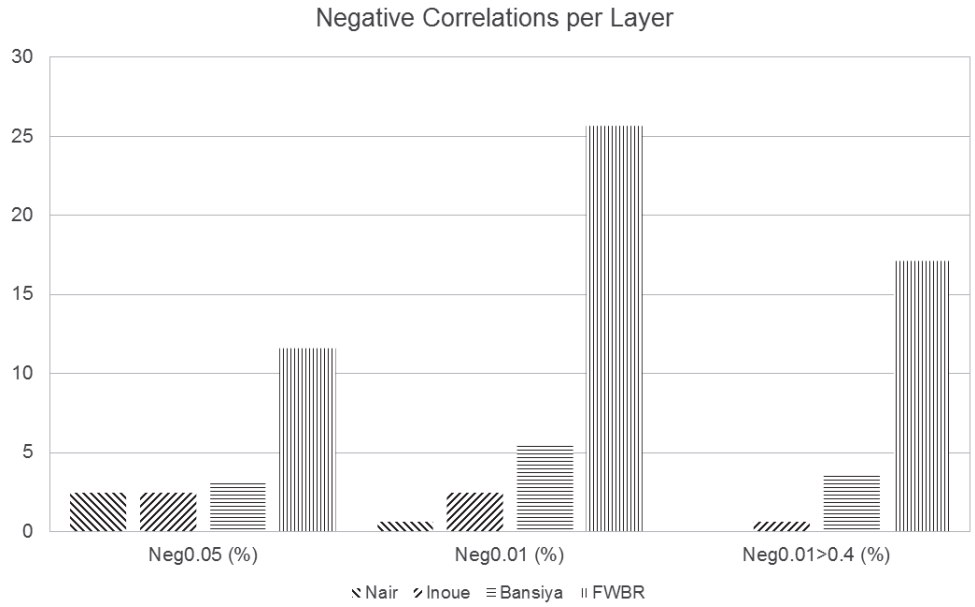

Figure 7: Percentages of negative correlations per layer and per project to the reuse metrics

in this test as can be seen from Table 11.

4. Analysis per project and around major project types

In our comparative assessment of the proposed metric in Section 3, Tests 1 & 2 (Sections 3.1 and 3.2) were carried out using all the projects' classes. To avoid the danger of ignoring the differences that individual projects may have in our assessment we repeat in this Section Tests 1 & 2 for all the individual projects separately. Test 3 (Section 3.3) was already performed for each project separately.

Table 12 shows the results of the correlation of the proposed metric compared to the other three metrics with respect to the CDS. As can be seen the proposed metric outperforms the other metrics in all cases with the excep-

Table 11: Percentages of positive correlations per layer and per project to the reuse metrics

	Positive correlations significant at 0.05 level	**Positive correlations significant at 0.05 level (%)**	**Positive correlations significant at 0.01 level**	**Positive correlations significant at 0.01 level (%)**
Nair	14	8.54%	32	19.51%
Inoue	16	9.76%	31	18.90%
Bansiya	13	7.93%	16	9.76%
FWBR	4	2.44%	0	0.00%
Total Layers examined				164

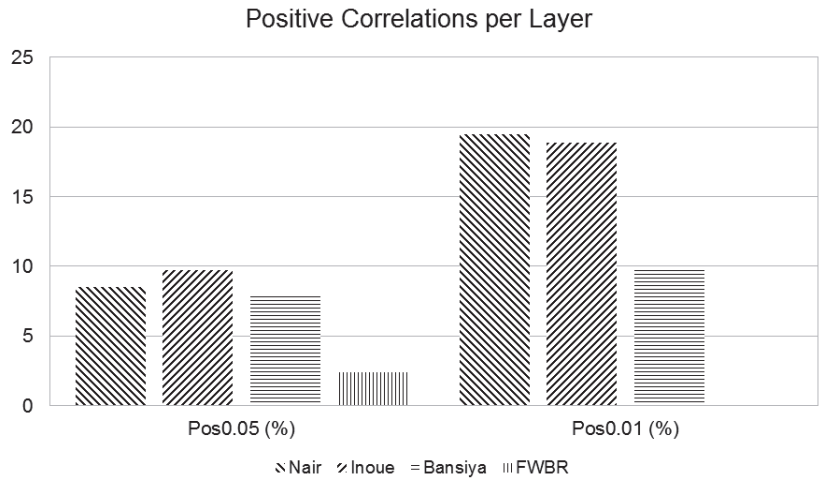

Figure 8: Percentages of positive correlations per layer and per project to the reuse metrics

tion of one project (Project 12 - FreeCS) in which the Inoue metric is slightly stronger.

Table 13 shows the results of the correlation of the proposed metric compared to the other three metrics with respect to the D-Layer and different partitions of the D-Layer. We remind the reader that classes higher at the layer structure of a project are usually less reusable and more application specific. A characteristic example of this are the user interface classes. On the contrary classes lower at the layer structure of the system are more reusable as for example generic business classes (e.g. a currency converter). The proposed metric outperformed the other proposed metrics in all 134 different partitions, with the one exception of the FreeCS project again in which the Inoue metric is slightly stronger. FreeCS has only four D-layers and therefore we could not decompose further in 5 layers or more. The correlation of the Inoue metric to D-Layer in FreeCS was $-.704$ whereas the correlation of the

Table 12: Spearman's ρ for the reuse metrics correlation to CDS per project

Project	Nair	Inoue	Bansiya	FWBR	No of Classes
1	.437**	-.366**	-.320**	**-.781****	447
2	.237**	-.488**	-.147**	**-.654****	493
3	.454**	-.321**	-.451**	**-.604****	1578
4	.271**	-.350**	-.258**	**-.506****	656
5	.658**	-.534**	-.475**	**-.864****	147
6	.145	-.107	-.468**	**-.521****	85
7	.262**	-.386**	-.539**	**-.820****	1100
8	.320**	-.149**	-.262**	**-.823****	2221
9	.306**	-.409**	-.432**	**-.774****	836
10	.344**	-.103**	-.035	**-.629****	2207
11	.359**	.210**	-.456**	**-.891****	155
12	.191*	**-.701****	-.368**	-.612**	141
13	.243**	-.441**	-.298**	**-.478****	406
14	.324**	-.255**	.097*	**-.779****	312
15	.174**	-.149**	-.155**	**-.643****	1987
16	.283*	-.119	-.204	**-.631****	58
17	.289**	-.289**	-.386**	**-.670****	520
18	.319**	-.382**	-.269**	**-.566****	508
19	.292**	-.264**	-.219**	**-.889****	172
20	.363**	-.223	-.489**	**-.718****	54
21	.231**	-.383**	-.412**	**-.639****	955
22	.433**	-.178**	-.074**	**-.738****	1759
23	.065	-.337**	-.244**	**-.520****	452
24	.291**	-.436**	-.386**	**-.591****	2745
25	.421**	-.146**	-.188**	**-.714****	335
26	.670**	-.662**	-.282*	**-.697****	56
27	.086*	-.410**	-.218**	**-.502****	377
28	.403**	-.619**	-.421**	**-.870****	167
29	.488**	-.266**	-.161**	**-.766****	846

**. Correlation is significant at the 0.01 level (1-tailed).

*. Correlation is significant at the 0.05 level (1-tailed).

proposed reuse metric was −.614. In Table 13 we report the actual values for four different projects using the original D-Layer and four different partitions of the D-Layers in 5, 6, 7 and 8 layers. The projects depicted are selected based on different number of classes. A small, medium, large and very large project are depicted, but as we mentioned already the results are similar for the rest of the analyzed projects with the exception of FreeCS. As can be seen from Table 13, the proposed metric demonstrates significant and strong correlations as expected whereas the Inoue metric outperforms in most cases the Bansiya proposed metric. Both these metrics also demonstrate negative correlations to the different layer partitions.

Table 13: Spearman's ρ for the reuse metrics correlation to different layer partitions per project

Project	Partition	Inoue	Bansiya	Nair	FWBR	No of Classes
19	D-Layer	-.268**	-.229**	.284**	**-.879****	172
	DLayer5	-.144*	-.285**	.265**	**-.781****	
	DLayer6	-.148*	-.284**	.263**	**-.781****	
	DLayer7	-.279**	-.173*	.307**	**-.854****	
	DLayer8	-.289**	-.180**	.301**	**-.847****	
17	D-Layer	-.319**	-.384**	.241**	**-.644****	520
	DLayer5	-.278**	-.367**	.202**	**-.598****	
	DLayer6	-.303**	-.379**	.223**	**-.624****	
	DLayer7	-.285**	-.377**	.244**	**-.631****	
	DLayer8	-.286**	-.378**	.244**	**-.631****	
21	D-Layer	-.419**	-.394**	.255**	**-.637****	955
	DLayer5	-.321**	-.330**	.284**	**-.602****	
	DLayer6	-.401**	-.374**	.258**	**-.628****	
	DLayer7	-.401**	-.374**	.259**	**-.628****	
	DLayer8	-.421**	-.390**	.249**	**-.629****	
10	D-Layer	-.191**	-.078**	.247**	**-.543****	2207
	DLayer5	-.229**	-.059**	.224**	**-.475****	
	DLayer6	-.177**	-.088**	.222**	**-.497****	
	DLayer7	-.193**	-.093**	.214**	**-.499****	
	DLayer8	-.191**	-.092**	.215**	**-.498****	

**. Correlation is significant at the 0.01 level (1-tailed).

*. Correlation is significant at the 0.05 level (1-tailed).

Another important distinction has to do with the size of the projects. Size

is related to many different facets. Larger projects are more complicated and involve many developers. Also usually size relates to the maturity of a project or more specifically to the number of years devoted to its development. We considered therefore important to repeat the regression analysis that was performed in Sec. 2 to three different size categories and detect differences in the influence that the CK metrics may have in the reusability assessment. In order to carry out this size-specific assessment we separated the projects to small projects with fewer than 500 classes, medium projects with classes between 500 (inclusive) and $1,500$ and large with $1,500$ classes or more. The regression analysis again was carried out in a similar way as we did for all the projects together in Sec. 2 so that the results are comparable. We are interested in determining the way that the size of the project affects the coefficients of the CK metrics (i.e. their contribution and direction) and to what extent the calibrated metrics improve the assessment of the originally proposed metric.

According to the methodology followed also in Sec. 2, we first excluded the outliers from individual categories as suggested in [34] and then we performed a forward regression analysis of the logarithms of the CK metrics to the D-Layer. The different coefficients per project category are listed in Table 14. As can be seen CBO is the most negative factor to reuse regardless of project category and LCOM is the least influential factor. In the normal-scale project category is also worth mentioning that NOC is excluded from the coefficients. The second most influential factor to reuse for small and normal-scale projects is DIT as it is in the proposed metric. However in large projects DIT is replaced by RFC as the second most influential factor

preventing reuse. A possible explanation for this is that in large scale projects developers start to notice opportunities for code reuse in domain specific layers and create interesting domain specific class hierarchies. Perhaps this is towards the creation of a domain-specific framework for example, that will enable the project to extend including new features with ease. Therefore classes with higher DIT values start to appear in lower project layers as well and DIT consequently is not specific to higher layers anymore. On the contrary the usage of GUI frameworks, such as the Swing framework for Java, is probably responsible for large DIT values in even the smallest projects in higher UI layers, since developers extend deep GUI hierarchies to provide their own application GUI. This is indeed consistent with our initial view, mentioned in Table 2, that DIT can be viewed from different perspectives. However DIT remains one of the most preventing factors to white box reuse even for large scale projects.

Using the coefficients for the three different categories we can formulate three different equations similar to Eq. 2 for the proposed metric $FWBR$. The three different calibrated metrics for small ($FWBR_S$), medium ($FWBR_M$) and large ($FWBR_L$) projects are given in Equations 8, 9 and 10 respectively.

Table 14: Regression analysis results per project category

Category	Predictor	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
Small	$log(WMC+1)$	.067	.334	.012	.201	.841
	$log(DIT+1)$	3.112	.230	.320	13.521	.000
	$log(NOC+1)$	-.132	.430	-.003	-.307	.759
	$log(CBO+1)$	5.810	.177	.779	32.744	.000
	$log(RFC+1)$	-1.012	.237	-.260	-4.277	.000
	$log(LCOM+1)$	-.041	.105	-.009	-.391	.696
Normal	$log(CBO+1)$	9.023	.225	.846	40.180	.000
	$log(DIT+1)$	3.945	.292	.259	13.514	.000
	$log(WMC+1)$	-.678	.413	-.067	-1.641	.101
	$log(RFC+1)$	-.826	.315	-.127	-2.621	.009
	$log(LCOM+1)$	-.378	.148	-.040	-2.563	.010
Large	$log(CBO+1)$	8.151	.157	.696	51.845	.000
	$log(DIT+1)$	1.431	.205	.092	6.971	.000
	$log(LCOM+1)$	-.191	.094	-.022	-2.018	.044
	$log(RFC+1)$	2.501	.222	.375	11.246	.000
	$log(WMC+1)$	-2.788	.296	-.279	-9.427	.000
	$log(NOC+1)$	-1.242	.465	-.015	-2.673	.008

$$
\begin{aligned}
FWBR_S = -1 \times (&5.810 \times \log(CBO+1) \\
&+ 3.112 \times \log(DIT+1) \\
&+ 0.067 \times \log(WMC+1) \\
&- 1.012 \times \log(RFC+1) \\
&- 0.041 \times \log(LCOM+1) \\
&- 0.132 \times \log(NOC+1))
\end{aligned}
\tag{8}
$$

$$
\begin{aligned}
FWBR_N = -1 \times (&9.023 \times \log(CBO + 1) \\
&+ 3.945 \times \log(DIT + 1) \\
&- 0.678 \times \log(WMC + 1) \\
&- 0.826 \times \log(RFC + 1) \\
&- 0.378 \times \log(LCOM + 1))
\end{aligned} \tag{9}
$$

$$
\begin{aligned}
FWBR_L = -1 \times (&8.151 \times \log(CBO + 1) \\
&+ 1.431 \times \log(DIT + 1) \\
&- 2.788 \times \log(WMC + 1) \\
&+ 2.501 \times \log(RFC + 1) \\
&- 0.191 \times \log(LCOM + 1) \\
&- 1.242 \times \log(NOC + 1))
\end{aligned} \tag{10}
$$

To quantify further the benefit that we can expect by calibrating the proposed metric to different size categories we examined also how many classes change reusability 'categories'. Since the actual coefficients are different it is not interesting to examine in how many classes the metric value changed since the actual absolute value of the metric will almost certainly be different. Metrics however are indications of the reusability of classes and as suggested in [21] we can separate the metric value range in four different categories set by three different thresholds using the average (AVG) and standard deviation (STDEV) of the metric. The three thresholds are:

1. *Lower Margin*: $AVG(Metric) - STDEV(Metric)$

2. *Higher Margin*: $AVG(Metric) + STDEV(Metric)$
3. *Very High Margin*: $(AVG(Metric) + STDEV(Metric)) \times 1.5$

Classes which fall below the lower margin are considered having low reusability, classes between the lower and higher margins have normal reusability, classes between the higher and very high margins have high reusability and classes above the very high margin have very high reusability. We calculated the reusability of each class with the four different metrics: (a) the original metric ($FWBR$), (b) the calibrated metric for the small size project category ($FWBR_S$), (c) the calibrated metric for the normal size project category ($FWBR_N$), and (d) the calibrated metric for the large size project category ($FWBR_L$). Then we calculated the categories according to the aforementioned margins and classified each class to one of the aforementioned categories using each metric separately. Finally we examined for each different project category separately in how many cases (i.e. classes) there was an alteration to the classification of the class using the respective calibrated metric. The results of this examination are the following:

1. For the small project category (i.e. projects with fewer than 500 classes) the classification of classes using the original proposed metric $FWBR$ and the calibrated reuse metric $FWBR_S$ results in the same classification to the four reuse categories in 91% of the cases whereas 9% of the cases change reuse category. Therefore assuming that classes that change category are classified more accurately with the calibrated metric the maximum improvement that we can anticipate in our sample using the calibrated metric is 9%.
2. Similarly for the normal project category (i.e. projects between 500

class (inclusive) and $1,500$ classes) the classes that changed classification using the calibrated metric $FWBR_N$ compared to the classification using $FWBR$ are 6% whereas 94% remained in the same category giving a maximum possible classification improvement of 6% in our sample.

3. Finally for the large project category (i.e. projects with $1,500$ classes or more) the calibrated metric $FWBR_L$ classified classes differently than $FWBR$ in 5% of the cases with 95% of the classes remaining in the same category.

These results indicate that the proposed metric is probably sufficient as an indication for most cases and that the calibration process brings marginal improvements. However, the method that we used to calibrate and compare the metrics to different project sizes can also be used with different categorizations. For example if a company concentrates in a specific application domain that is expected to have significantly different reuse categories than the projects used in our analysis then perhaps a similar process can be followed to derive a domain-specific reuse metric by analyzing only projects that belong to this specific application domain.

5. Threats to validity

Conclusion validity in our study refers to the existence of a relationship between the design complexity metrics and white-box reuse. We have established a relationship between the design complexity metrics and D-Layers as well as CDS. These metrics are associated to white-box reuse. Furthermore the relationship of design complexity metrics and reusability is also discussed

in a number of works that we examined extensively in Sec. 3 and others that we discuss in Sec. 7.

Internal validity in our study refers to establishing causality between the design complexity metrics and white-box reuse. As we have said our FWBR metrics is *facilitative* to white-box reuse. It must be combined with other factors such as the subjective judgment of the reuser regarding the usefulness of a class for reuse purposes. In this work as we have already mentioned, the emphasis was on the technical characteristics that can be considered necessary factors for the white-box reuse of a class. The current study will be complemented in the future as we mention in Sec. 8 with other factors that will extend this work beyond the technical level, to also include users' experience using the COPE tool and the proposed metric specifically for the component extraction activity.

Regarding *construct validity* the use of non-parametric tests minimized the assumptions about the data. We have used in this work the original definitions of the Chidamber and Kemerer metrics as presented in Table 2. Some tools support however slightly different definitions of these metrics. Users of our proposed metric should be careful in using tools that support the original definitions. Regarding the exclusion of outliers we believe that they were excluded correctly because they are not representative of the majority of the classes in a system and they would heavily influence the regression results. The problem is that classes which are extremely complex exist in every system for reasons that are not relevant for the current work. Usually designers and programmers are aware of these problematic cases. The reusability of these particular cases is rather low but this is something acceptable by the

programmers because they do not consider these classes as candidates for reuse (i.e. they are very application-specific classes). In our work we have tried to limit our study in classes that are potentially reusable since these are the subject under consideration.

In relation to *external validity* we have used only Java open source projects. So the results are valid for this category of projects. The results are probably generalizable to other Object-Oriented programming languages similar to Java (e.g. C#) however they can be inapplicable for languages with different characteristics (e.g. scripting languages) due to different programming idioms. Furthermore, some application domains are not represented in our sample. For example real-time systems are not present in this study. However as already discussed in Sec. 4, if a company concentrates in a specific application domain that is expected to have significantly different reuse categories than the projects used in our analysis then perhaps a similar process as the one presented in Sec. 4, can be followed to derive a domain-specific reuse metric by analyzing only projects that belong to this specific application domain. Sec. 4 discusses extensively the project types for which our work applies and provides calibration approaches where necessary.

6. Discussion

Components are easier to reuse than frameworks, although frameworks provide more flexibility. Reusing source code is even harder. So the desirable form of reuse, in the sense that it is the easier for the programmers, is component black box reuse. Domain specific coarse grained software components are usually created from frameworks and frameworks from source

code. The authors in [20] for example provide a number of patterns to be used in the process of creating a framework from source code. It is important therefore, to be able to assess the reusability of source code as a first step in the direction of a process that will eventually lead to a library of reusable components and this is where the current work is mostly relevant. Current work does not attempt to contradict the widely accepted view that black-box reuse is preferable [4], but rather it tries to provide a tool for getting there, that can be used in conjunction with other proposed techniques such as refactoring [25, 16], and patterns [20]. It also tries to discriminate the different types of reuse and warn against providing metrics without specifying the relevance of the metric to its respective reuse type (i.e. black-box, gray-box and white-box reuse). This vagueness in specifying precisely the context of a reuse metric can be confusing and harmful.

Our work aligns with other works' conclusion [29, 18] that coupling is a very significant limiting factor for the white-box reuse of classes. It seems extremely important therefore to have dependency control in a project. Packages are a form of dependency control and a number of patterns and metrics have been developed for the creation of package structures that are easier to maintain [23]. However, the extraction of components from an existing system requires extracting subsets of packages' classes, possibly from many different packages. The package dependency control therefore, seems more relevant to corrective, adaptive and perfective maintenance [31] than it is for a componentization activity, since component classes in the general case crosscut the package structure. Although package-based dependency control can help, it would be even more helpful to have a mechanism to restrict de-

pendencies vertically, i.e. to have some form of feature isolation. Existing module systems for example, such as the NetBeans Rich Client Platform [5] and Eclipse Rich Client Platform [24], promote applications that are composed of components. In these platforms features embodied in components can be pluggable in a top level container at the user interface level. Since components communicate through carefully designed interfaces, the classes inside each component are not left free to use all the classes below them in the layered architecture of the system, but only classes that are included in the same component. This is expected to have a significant effect in the CDS metric discussed in this work, since classes at higher layers will have fewer dependencies.

Our approach in the current work was to validate the proposed metric using other metrics that can be computed automatically from source code static analysis and are logically related to ease of understanding, adapting, testing etc. source code. The two metrics used were the number of required classes for reusing a class and the layer of a class. Indeed reusing a class requires other classes and therefore the larger the number of these classes the more difficult will be to reuse the class. As we saw these two metrics are strongly and significantly correlated (Table 3) as expected. This approach was selected because it leads in repeatable results which are a major problem with current reuse metrics as mentioned in a recent software components reusability assessment survey [17]. In this survey the Inoue et al. [19] and Washizaki et al. [33] approaches are identified as the only proposals "validated with industry-level observational studies", while the authors of this survey also observe that the majority of the proposals discussed in their sur-

vey "were not validated at all". The results of current work besides being repeatable, are also based in an extensive analysis of 21,775 classes of 29 OSS projects. However a possible criticism in our validation approach is that it ignores external reusers and important aspects for external reusers such as the documentation and the business value of the reusable components. As we already mentioned interface metrics and documentation are related more with black box and gray box reuse whereas the proposed metric is mostly targeting white box reuse as a starting point for evolving source code to frameworks and components. Consequently we concentrated to important characteristics to white-box reuse. Although we do not discuss the business value of the produced components using the FWBR metric, the proposed metric did quite well in the Third test described in Sec. 3.3 compared to other metrics. This signifies that it is a valid metric for classes with similar roles in a system's architecture. To the extent that classes play similar roles it can be argued that share to some extent the same business value. In a follow-up study we plan to investigate the question of 'business value' using empirical methods including reusers' opinions.

7. Related work

Besides the works we already examined [3, 19, 27] we review in this Section a number of related works that used some subsets or modified versions of metrics in [9] to determine the reusability of classes or more coarse-grained software units (e.g. components, libraries etc.). Also we review some works that used their own metrics developed from scratch. For each work we attempt to establish a relation to the proposed approach in this work.

In [4] the author concentrates on object-oriented metrics commonly associated with reusability. This work describes two experiments: one quick experiment in which two developers were asked to develop a highly reusable and a very application-specific component respectively and their results were compared with the metrics list. The second experiment involved applying the metrics list to well-known OO libraries from several OO languages (assumed to be highly reusable). The experiments demonstrated that some of the commonly cited as important factors for reusability, in reality do not affect a component's reusability: "For instance, it was revealed that method size (lines of code), the number of methods and the number of attributes do not appear to have any bearing on how reusable the class is". On the contrary interfacial metrics such as the coupling of a class with other classes, documentation, correctness and naming choices are very important metrics strongly affecting reusability. This suggests that a black-box view for component reusability is the most important, ignoring internal factors. The author then presents the most influential metrics for the reusability of classes and combines these metrics in a single formula (a new metric called "Reusabity for a class Rc") that can be used as a reusability oracle when developers consider including a class in a reuse repository. An unreusable class has a reusabilty for a class metric value of less than 5, whereas a reusable class has a value more than 10. Compared to current work, this work concentrates more in the black-box reuse since it includes aspects such as the documentation. However since the components examined were classes it also considers coupling a very important factor.

In [29] the author uses an empirical study of NASA software reuse to de-

termine the factors that characterize successful reuse in large-scale systems. The study follows the goal-question-metric approach to characterize software reuse at the project level, module design level, module implementation level and reuse and module faults and changes. The modules examined were subroutines, utility functions, main programs, macros and block data. The interpretation of the statistical analyses demonstrated that modules that were reused without revisions were mainly small, well-documented with simple interfaces and little input-output processing. Furthermore they were lower in the invocation hierarchy ("terminal nodes") which resulted in fewer interactions with other modules and high interactions with utility modules (low level general purpose modules). Also the faults in modules reused without revisions were few and consequently the fault-correction effort was small. Our results to large extent coincide with the empirical findings in [29], albeit for OO software.

In [2] the authors examine ease of reuse for OO class libraries of Java and Eiffel. To determine the ease of reuse they examine the number of public methods of a class (a modified version of WMC with method complexity 1 that accounts only for public methods). The reason for including public methods is that during (black box) reuse only the interface matters. A large number of methods indicates according to the authors a more application-specific and therefore less reusable class. However larger WMC can also be beneficial to reusers because it signifies components providing more services. Depth of Inheritance (DIT) is also used. Larger values indicate that the re-user needs to examine a larger number of ancestor classes to successfully reuse a class. Also larger DIT values indicate more specificity and less gen-

eralization. This work differs from ours since it examines reusable libraries which are already formed components or frameworks. Therefore this work is more relevant to black-box reuse.

In [8] the authors also consider OSS reusability based on metrics. They use the metrics proposed by [23]. This work is concerned with folder reuse of OSS repositories. It calculates the instability of a folder as $I = \frac{Ce}{(Ce+Ca)}$ where Ce is the efferent coupling (folders depending on (i.e. directly calling) this folder) and Ca is the afferent coupling (folders this folder depends upon). Stability is the complement of instability ($S = 1 - I$). They provide a case study (XMMS) of their approach. This work differs than ours in that it considers packages and therefore examines more ready-to-reuse solutions than classes.

In [33] the authors describe a metrics suite suitable for black box components that can be used to access the reusability of a component without its source code. The source code unavailability makes the use of other popular metric suites, such as the metrics used in our work [9], unsuitable since they are mostly based in source code analysis. The metrics suite proposed is explored in the context of the Java Beans component technology; its use however is also applicable to other contexts. Washizaki et al. approach is more concerned with black-box reuse of a specific technology (i.e. Java Beans) and therefore is not directly related to our work.

The work in [1] proposes two metrics in relation to templates used in an object-oriented system. The first metric is the Function Template Factor (FTF) and the second the Class Template Factor (CTF). These metrics can be used with O-O languages that provide template (generic) functions and

classes such as C++ and more recently Java and C#. FTF is the ratio of template functions to the total number of functions in the system. Similarly CTF is the ratio of template classes to the total number of classes in the system. In relation to our work, this work considers template functions and classes but does not incorporate other characteristics of classes.

A number of European projects similar to OPEN-SME, which is the context of this work, explored the question of OSS quality in general and reusability in particular including SQO-OSS [28] and QualiPSo [12]. The difference of OPEN-SME is that it is narrower since the reuse service is provided in the context of specific domains, and deeper since it attempts to provide more advanced reuse services in these domains including component extraction and certification. The facilitation of the reusability assessment described in this work is one of the fundamental requirements of such an approach to reuse services.

8. Conclusions and future research directions

In this work we proposed a new facilitative metric for white-box reuse of Object-Oriented source code. This type of reuse can be useful in many settings including component extraction and re-architecting of existing systems. The proposed metric can also be used during the iterative development of a new project where developers can detect reuse problems of software components that are candidates for reuse in other applications. We evaluated the proposed metric by comparing it to three other metrics from the literature, and demonstrated significant advantages in relation to white-box reuse.

Concerning future extensions of current work, in the context of the OPEN-

SME FP7 project we will test the COPE tool for the extraction or reusable components in two planned user trials. During these trials we plan to extend this work beyond the technical level described here, to include also users' experience using the tool and the proposed metric specifically for the component extraction activity. Also we will contrast the $FWBR$ metric based component extraction with other forms of extraction (e.g. pattern-based component extraction) that are also developed in the context of the project.

The proposed metric can be useful in highlighting the architectural layers of applications. Current work in our team attempts to identify architecture layers using metrics [11, 10]. Specifically for the widely used layered architecture style, the proposed metric can be used to distinguish between the layers of the architecture of an existing system. This is another research direction we plan to explore in the context of Software Architecture Reconstruction (SAR) [13].

This work concentrated mainly on the measurable characteristics of the source code to derive an assessment that could be used in the context of white-box reuse. However other factors relating to the comprehensibility of the source code, such as the coding standards followed, can also be important for white-box reuse. It will be an interesting extension of this work, to examine such quality factors and compare the findings with the findings of this work.

References

[1] Aggarwal, K., Singh, Y., Kaur, A., Malhotra, R., 2005. Software reuse metrics for Object-Oriented systems. In: Software Engineering Re-

search, Management and Applications, ACIS International Conference on. IEEE Computer Society, pp. 48–55.

[2] Araban, S., Sajeev, A., 2006. Reusability analysis of four standard Object-Oriented class libraries. In: Software Engineering Research and Applications. Vol. LNCS 3647/2006. Springer, pp. 171–186.

[3] Bansiya, J., Davis, C., jan 2002. A hierarchical model for object-oriented design quality assessment. IEEE Transactions on Software Engineering 28 (1), 4–17.

[4] Barnard, J., Mar. 1998. A new reusability metric for object-oriented software. Software Quality Journal 7 (1), 35–50.

[5] Boudreau, T., Tulach, J., Wielenga, G., May 2007. Rich Client Programming: Plugging into the NetBeans Platform. Prentice Hall.

[6] Buschmann, F., Meunier, R., Rohnert, H., Sornmerlad, P., Stal, M., 1996. Pattern-Oriented Software Architecture: A System of Patterns. Wiley.

[7] Campbell, C., Grieskamp, W., Nachmanson, L., Schulte, W., Tillmann, N., Veanes, M., May 2005. Model-Based testing of Object-Oriented reactive systems with spec explorer. Technical Report MSR-TR-2005-59, Microsoft Research.

[8] Capiluppi, A., Boldyreff, C., May 2007. Coupling patterns in the effective reuse of open source software. In: First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07). IEEE, p. 9.

[9] Chidamber, S., Kemerer, C., Jun. 1994. A metrics suite for object oriented design. IEEE Transactions on Software Engineering 20 (6), 476–493.

[10] Constantinou, E., Kakarontzas, G., Stamelos, I., 2011. Open source software: How can design metrics facilitate architecture recovery? CoRR abs/1110.1992.

[11] Constantinou, E., Kakarontzas, G., Stamelos, I., 2011. Towards open source software system architecture recovery using design metrics. In: Panhellenic Conference on Informatics. IEEE, pp. 166–170.

[12] del Bianco, V., Lavazza, L., Morasca, S., Taibi, D., Tosi, D., 2010. The qualipso approach to oss product quality evaluation. In: Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development. FLOSS '10. ACM, pp. 23–28.

[13] Ducasse, S., Pollet, D., July 2009. Software architecture reconstruction: A process-oriented taxonomy. IEEE Transactions on Software Engineering 35, 573–591.

[14] Elmer, F.-J., 2011. Classycle: Analysing Tools for Java Class and Package Dependencies. `http://classycle.sourceforge.net`, [Online; accessed 27-November-2011].

[15] Field, A., Jun. 2009. Discovering Statistics Using SPSS, 3rd Edition. Sage Publications, Inc.

[16] Fowler, M., 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley.

[17] Goulão, M., e Abreu, F. B., 2011. An overview of metrics-based approaches to support software components reusability assessment. CoRR abs/1109.6802.

[18] Gui, G., Scott, P. D., September 2007. Ranking reusability of software components using coupling metrics. Journal of Systems and Software 80, 1450–1459.

[19] Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M., Kusumoto, S., march 2005. Ranking significance of software components based on use relations. IEEE Transactions on Software Engineering 31 (3), 213 – 225.

[20] Johnson, R. E., Foote, B., June/July 1988. Designing reusable classes. Journal of Object-Oriented Programming 1 (2), 22–35.

[21] Lanza, M., Marinescu, R., 2010. Object-Oriented Metrics in Practice. Springer.

[22] Larman, C., 2004. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd Edition. Prentice Hall.

[23] Martin, R. C., Martin, M., Jul. 2006. Agile Principles, Patterns, and Practices in C#, 1st Edition. Prentice Hall.

[24] McAffer, J., Lemieux, J., Aniszczyk, C., May 2010. Eclipse Rich Client Platform, 2nd Edition. Addison-Wesley Professional.

[25] Mens, T., Tourwe, T., Feb 2004. A survey of software refactoring. IEEE Transactions on Software Engineering 30 (2), 126–139.

[26] Microsoft Patterns and Practices Team, 2009. Microsoft Application Architecture Guide, 2nd Edition. Microsoft Press.

[27] Nair, T. G., Selvarani, R., January 2010. Estimation of software reusability: an engineering approach. SIGSOFT Softw. Eng. Notes 35, 1–6.

[28] Samoladas, I., Gousios, G., Spinellis, D., Stamelos, I., 2008. The sqo-oss quality model: Measurement based open source software evaluation. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (Eds.), Open Source Development, Communities and Quality. Vol. 275 of IFIP International Federation for Information Processing. Springer Boston, pp. 237–248.

[29] Selby, R., Jun. 2005. Enabling reuse-based software development of large-scale systems. IEEE Transactions on Software Engineering 31 (6), 495–510.

[30] Spinellis, D., July/August 2005. Tool writing: A forgotten art? IEEE Software 22 (24), 9–11.

[31] Swanson, E. B., 1976. The dimensions of maintenance. In: Proceedings of the 2nd international conference on Software engineering. ICSE '76. IEEE Computer Society Press, pp. 492–497.

[32] Tarjan, R., 1972. Depth-first search and linear graph algorithms. SIAM J. on Computing 1 (2), 146–160.

[33] Washizaki, H., Yamamoto, H., Fukazawa, Y., 2003. A metrics suite for measuring reusability of software components. In: Proceedings of the 9th International Symposium on Software Metrics. IEEE Computer Society, pp. 211–223.

[34] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, A., 2000. Experimentation in Software Engineering: an Introduction. Kluwer Academic Publishers, Norwell, MA, USA.