The Journal of Systems and Software 86 (2013) 349–366

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Layer assessment of object-oriented software: A metric facilitating white-box reuse☆

George Kakarontzas[a,b,*], Eleni Constantinou[a], Apostolos Ampatzoglou[a], Ioannis 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

ARTICLE INFO

Article history:
Received 16 December 2011
Received in revised form 10 August 2012
Accepted 21 August 2012
Available online 29 August 2012

Keywords:
Software reuse
Object-oriented metrics
Software metrics

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 also in providing assistance during re-architecting and componentization activities of existing systems.

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 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 (Chidamber and Kemerer, 1994) 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 (Buschmann et al., 1996) are not expected to share the same

☆ 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 at: Department of Computer Science and Telecommunications, T.E.I. of Larissa, 41110 Larissa, Greece.
E-mail addresses: gkakaron@teilar.gr (G. Kakarontzas), econst@csd.auth.gr (E. Constantinou), apamp@csd.auth.gr (A. Ampatzoglou), stamelos@csd.auth.gr (I. Stamelos).

[1] http://opensme.eu.

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.08.041

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 Section 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 Section 3 we compare our proposed metric with three other proposed reuse metrics from the literature. In Section 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 Section 5 we discuss threats to validity. Then, in Section 6 we discuss the results and findings of this study and in Section 7 we discuss related work. Finally in Section 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 (Chidamber and Kemerer, 1994), (b) the layer of each class as reported after condensing the cyclic dependencies of the class dependency graph using the Tarjan algorithm (Tarjan, 1972), 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.

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	21,775	

2.1. Metrics used

2.1.1. Chidamber and Kemerer metrics

The Chidamber and Kemerer metrics (Chidamber and Kemerer, 1994) were collected using the CKJM tool (Spinellis, 2005). 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 Chidamber and Kemerer (1994) 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 Chidamber and Kemerer (1994) 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 facilitative metric for white-box reuse.

2.1.2. The D-layer metric

The Classycle analyzer tool (Elmer, 2011) 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 (Tarjan, 1972). Next, according to SCCs calls, the graph is condensated to an acyclic digraph of SCCs, from which the layers are extracted (Fig. 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