Assessing the Evolution of Quality in Java Libraries

Theodore Chaikalis
Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
chaikalis@uom.gr

Alexander Chatzigeorgiou
Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
achat@uom.gr

Apostolos Ampatzoglou
Department of Mathematics and Computer Science, University of Groningen, Groningen, The Netherlands
a.ampatzoglou@rug.nl

Ignatios Deligiannis
Department of Information Technology Technological Education Institute, Thessaloniki, Greece
ignatios@it.teithe.gr

ABSTRACT
Libraries are increasingly employed in software practice to speed up the development process by reusing available and tested components. Software systems, that are available as libraries, are expected to be well-designed, because they have to adhere to specific principles, in order to accommodate the needs of multiple clients in a robust and stable way. Considering that most software libraries are continuously upgraded, in this paper we investigate the evolution of their quality over time. In particular, we perform a systematic case study to assess whether quality, in terms of three software metrics (CBO, LCOM, WMC), exhibits clear trends during the history of twenty analyzed libraries. The findings indicate that the examined software libraries can be considered as stable software projects in terms of quality, in the sense that in contrast to the general belief about software aging, their quality does not degrade over time.

Categories and Subject Descriptors
• Software and its engineering ~ Software creation and management • Software and its engineering ~ Software evolution • Software and its engineering ~ Maintaining software • *Software and its engineering ~ Object oriented development*

Keywords
Software evolution analysis; case studies, software quality.

1. INTRODUCTION
A software library can be defined as a collection of software modules for supporting programming through a well-established Application Programming Interface (API) [4]. Beyond code, libraries entail a specified set of rules and conventions that should be applied for accessing the offered functionality. Libraries are intended for broad employment by numerous clients that extend their functionality by reusing already available code (either in the form of source code or as compiled modules).

Based on their original purpose of use, libraries are considered well-designed pieces of code, which adhere to software design principles. The main rational beneath this belief is that libraries are intended to support a large number of clients, and for this reason their external interface should a) allow seamless integration with client code, b) remain constant over successive versions so as not to 'break' client code and c) be extensible to allow clients to define their own specific implementations [18]. These requirements impose specific constraints on the internal software development practices and usually promote a clean, rigid and robust software architecture [18].

As any other software product, libraries are continuously evolving by releasing new versions that offer enhanced functionality or improved performance. Along the evolution of software systems, the general belief is that their quality degrades over time due to the need to accommodate several requirements under significant time pressure. This phenomenon has been extensively studied in the literature of software engineering and is known under different names such as *software aging* [15] or accumulation of *technical debt* [1]. Considering the special characteristics of software libraries it would be worth exploring whether software libraries suffer from the same symptom.

In this paper we aim at assessing the evolution of quality in well-known libraries. To this end, we conducted a case study in which we evaluated trends in the evolution of three typical object-oriented metrics on 20 OSS libraries. The existence of clear trends in the evolution of quality has been assessed by appropriate statistical tests. The results open up opportunities for discussing whether it is worth to transfer the principles underlying the design of software libraries to other types of software as well.

The rest of the paper is organized as follows. In section 2 we briefly discuss related work on software evolution analysis and assessment of quality in libraries. The design of the case study is described in Section 3 while the results are presented and discussed in Section 4. Threats to validity are listed in section 5. Finally, we conclude in Section 6.

2. RELATED WORK
The analysis of evolutionary trends in the history of software projects has been extensively studied during the last decade in the literature of Software Engineering. The foundations for this area have been laid by M. M. Lehman in the 70's who defined and later enhanced the so-called *laws of software evolution* [12]. A good overview of the field as well as trends in software evolution research can be found in the book edited by Mens and Demeyer [14]. Numerous types of analyses and statistical tools have been applied to investigate all aspects of software evolution offering answers to practical questions relevant to software practice, as well as, interesting insights related to phenomena governing software evolution [3], [5], [10].

BCI'15, September 02-04, 2015, Craiova, Romania.

DOI: http://dx.doi.org/10.1145/2801081.2801097

The challenges in designing stable and reliable libraries have been addressed and systematically documented in the form of good practices for API design [4], [18]. Raemaekers et al. [16] evaluated the stability of third party libraries in terms of method removals, changes in the implementation and method additions. McDonnell et al. [13] studied the pace at which libraries in the android ecosystem evolve along with the client adoption, observing that clients usually do not catch up with the API evolution. API changes of four frameworks and one library have been studied by Dig and Johnson [9] discovering that API-breaking changes indeed occur during the history of libraries.

With respect to the design quality of libraries, an application of an operations research methodology (Data Envelopment Analysis) revealed that libraries exhibit superior quality compared to software applications [7]. However to the best of our knowledge, no case study has been performed to formally assess trends in the *evolution* of quality in libraries by means of metrics.

3. CASE STUDY DESIGN

The design of the case study regarding the evolution of software libraries will be described briefly due to space limitations according to the guidelines proposed by Runeson et al. [17]

Objective and Research Questions

Using the Goal-Question-Metric (GQM) formulation [2], the goal of this study can be expressed as: *"to **analyze** successive versions of software libraries **for the purpose of** evaluating the evolution of their quality **with respect to** the trend of basic object-oriented design metrics **from the perspective of** researchers **in the context of** 20 open source libraries.* Based on this goal the research question under investigation is:

RQ: Do software libraries exhibit an observable trend in the evolution of their quality?

Selection of Cases

To ensure the selection of well-known, mature and reliable open source libraries as cases for our study we employed the following approach: We have selected open source software systems (applications) which a) are written in Java, b) evolved over a number of versions, c) have a large developer and user community, and d) are among the most downloaded products in their domain. With these criteria we aimed at collecting a set of mature and reliable software products. Then we extracted the libraries on which these systems rely. The corresponding assumption is that since the selected applications fulfill certain criteria, the corresponding libraries will meet similar standards. The selected libraries are listed in Table I.

Data Collection

For each version of the analyzed projects we obtained the following measures from the Metrics Suite proposed by Chidamber and Kemerer [8]. Although these metrics are among the oldest in the literature of object-oriented design they have been repeatedly applied to assess software design quality and their interpretation is straightforward [11]. From the metrics that are offered in the Chidamber and Kemerer metrics suite, we picked one metric from each quality property, i.e., coupling, cohesion and complexity, as follows:

CBO – *Coupling Between Objects*: The number of other classes to which a class is coupled.

LCOM- *Lack of Cohesion in Methods*: Quantification of lack of cohesion based on the number of cohesive and non-cohesive method pairs.

WMC- *Weighted methods per class*: The sum of the complexities of a class' methods.

The aggregation function of all metrics to obtain values at the system level has been set to *average*.

The extracted data form time series for each metric and for each project. The evolution of metrics for each project has been obtained using the SEAgle platform [6] developed by the authors. SEAgle enables effortless software evolution analysis where the user provides only the git repository of the project that he/she wishes to analyze. In response, the platform provides a wide spectrum of results concerning the analyzed project, through a web interface. Analyses includes various metrics, i.e., metrics concerning repository activity, as well as, metrics related to the object-oriented structure of the systems. All metrics are presented in the form of a series of values over the successive versions that have been analyzed. SEAgle is accessible as a web application and as a RESTful Web Service.

Table 1. Analyzed libraries

#	Name	Description	Versions
1	ant	Library for the building of Java applications	9
2	antlr4	Parser generator for reading, processing, executing, or translating structured text or binary files	6
3	axis2	Apache implementation of SOAP for Web Services	10
4	checkstyle	Tool to help programmers write Java code that adheres to a coding standard.	30
5	commons-io	Library of utilities to assist with developing IO functionality.	8
6	commons-lang	Library that provides extra methods for the manipulation of Core Java Classes.	12
7	guava	Collection of core java libraries used by Google for their java-based projects.	17
8	hazelcast	Open Source In-Memory Data Grid	40
9	jackson-core	Core part of Jackson JSON Processor that defines Streaming API and basic shared abstractions	29
10	jackson-databind	General data-binding package for Jackson JSON Processor	31
11	joda-time	Replacement library for the Java date and time classes.	16
12	junit	A programmer-oriented testing framework for Java.	16
13	log4j	Logging library by the Apache Software Foundation.	73
14	mockito	Mocking framework for unit tests written in Java.	26
15	netty	Event-driven asynchronous network application framework	8
16	ognl	Object Graph Navigation Library	9
17	pdfbox	Library for the creation of new, and manipulation of existing PDF documents.	21
18	sisu	Implementation of JSR 330 (Context and Dependency Injection)	25
19	smack	Open Source Extensible Messaging and Presence Protocol Client Library written in Java	12
20	zookeeper	Librar to develop and maintain an open-source server which enables highly reliable distributed coordination	11

* Further information on the selected libraries can be found in SEAgle.

Data Analysis

In order to investigate the research question that has been set, we will perform a trend test on each time series. Trend analysis aims at determining whether the values of a series of temporal observations generally increase or decrease. In statistical terms a trend test assesses whether the probability distribution from which the analyzed values come from, has changed over time. The corresponding null hypothesis can be stated as:

H_0: *there is no trend in the evolution of the observed metric*

Thus, the goal of the statistical analysis is to accept or reject this null hypothesis. An established approach for conducting a trend test is to fit a linear function on the observed data (linear regression) and determine the slope of this trendline in case the corresponding p-value of the linear regression analysis implies a statistically significant result. However, linear regression is a parametric approach and a number of conditions have to be satisfied to be able to apply it. These assumptions include:

- Absence of significant outliers,
- independence of observations
- homoscedasticity and,
- approximately normally distributed residuals.

These assumption can be formally checked by appropriate statistical tests. For example, the independence of observations, i.e. that data exhibit little or no autocorrelation, can be tested with Durbin-Watson's test. After applying the relevant tests to our dataset (timeseries of metrics for the examined projects) we found that none of the cases could be fitted to linear regression models, since one or more of the preconditions were not met. Therefore, to provide robust statistical results we performed the Mann-Kendall non-parametric trend test which assesses whether there is a monotonic upward or downward trend of the independent variable (i.e. metric). This test does not impose the preconditions, especially with regard to the normal distribution of residuals. We calculated the corresponding statistic using the R language [20]. The dataset on which the statistical tests have been applied as well as the corresponding R scripts can be found in the accompanying web page [19].

For the cases where a trend is statistically evident we calculated the slope of the corresponding trendline. To enable the comparison of trends between different projects and metrics a scale invariant measure of slope should be extracted. To this end, we normalized the original data by dividing each value with the maximum value in the timeseries. Moreover, expressing the slope as a percentage, enables an intuitive interpretation of the steepness of observed trends.

Table 2. Trend tests and slopes for CBO, LCOM and WMC

Name	CBO			LCOM			WMC		
	Sig.	Trend	Slope	Sig.	Trend	Slope	Sig.	Trend	Slope
ant	0.009	↗	4.00%	0.348	→		0.602	→	
antlr	1.000	→		0.338	→		0.008	↗	1.47%
axis2	0.177	→		0.785	→		0.210	→	
checkstyle	0.629	→		0.068	→		0.000	↘	-0.53%
commons-io	0.462	→		0.221	→		0.806	→	
commons-lang	0.318	→		0.002	↗	0.96%	0.901	→	
guava	0.543	→		0.022	↘	-1.87%	0.692	→	
hazelcast	0.002	↗	0.62%	0.000	↘	-0.18%	0.000	↘	-0.60%
jackson-core	0.000	↗	0.09%	0.000	↗	0.23%	0.148	→	
jackson-databind	0.042	↘	-0.04%	0.000	↘	-0.17%	0.035	↘	-0.03%
joda-time	0.000	↗	1.42%	0.030	↘	-0.43%	0.000	↗	1.40%
junit	0.000	↘	-0.75%	0.377	→		0.000	↘	-0.76%
log4j	0.004	↘	-0.23%	0.966	→		0.563	→	
mockito	0.697	→		0.697	→		0.000	↘	-0.34%
netty	0.001	↗	2.02%	0.012	↗	1.70%	0.035	↗	2.50%
ognl	0.002	↘	-0.36%	0.004	↗	1.78%	0.529	→	
pdfbox	0.001	↘	-0.10%	0.000	↗	0.70%	0.000	↗	0.34%
sisu	0.006	↘	-2.2%	0.000	↘	-0.41%	0.001	↘	-2.51%
smack	1.000	→		0.136	→		0.782	→	
zookeeper	0.220	→		0.027	↘	-1.12%	0.462	→	

* Statistical significance level is set to 0.05

4. RESULTS AND DISCUSSION

The results concerning the statistical tests on whether metric time series exhibit trends or not are summarized in Table 2. The first column lists the project's name, while the rest of the columns summarize the findings for each of the three metrics. For each metric, the table reports the significance value of the Mann Kendall trend test. In case the corresponding sig. value is less than 0.05 the trend is considered statistically significant and in these cases a down/up pointing arrow implies an improving/deteriorating quality over time.

It should be noted that for the selected metrics, an improvement is reflected by a decrease in the metric values. For the cases where a trend cannot be determined based on the statistical test, we plotted a horizontal right pointing arrow. When a trend is present, the slope of the corresponding trendline is listed in the last column for each metric. As it can be observed, in about half of the cases no trend is present and thus no definite answer can be provided to the research question of this study. However, by focusing on the cases where the results imply stability or improvement the picture becomes more clear. In particular, in 16 of the 20 projects w.r.t. CBO, 15 of the projects w.r.t. LCOM and 16 of the projects w.r.t. WMC (in ~78% of the cases) metric values either remained stable or improved during the evolution of the libraries. Thus, one could claim that libraries indeed exhibit signs of resilient object-oriented design which in turn is reflected on non-deteriorating metric values. It is worth mentioning, that in several projects quality is improving over time, sometimes at a significant pace. For example, project *sisu* improves in terms of all three examined metrics by 2.2, 0.41 and 2.51 per cent, for CBO, LCOM and

WMC, respectively. On the other hand, there are limited cases where degradation is observable in more than one aspects of quality. An exception is project *netty* whose values deteriorate in all three metrics.

As a result, libraries not only exhibit a superior design quality when assessed statically (i.e. when analyzing individual versions as it has been performed in previous approaches [7]) but also perform well in terms of stability and robustness over time. Such findings imply that their development teams indeed strive for conformance to proper design principles resulting in high quality architectures. We believe that software design research and practice could benefit by focusing on such well-constructed libraries and export the knowledge and techniques reflected in their internal structure.

5. THREATS TO VALIDITY

As in any case study the findings suffer from threats to external validity in the sense that the conclusions reflect only the particular libraries which have been analyzed. However, we believe that this threat is partially mitigated by the inclusion of 20 libraries covering different domains. With respect to construct validity which is related to the degree by which the employed measures reflect the phenomenon under investigation (i.e. the quality of libraries), two threats arise from the selection of projects and three particular metrics. On the one hand, the characterization of a software system as library cannot be absolute, in the sense that some libraries act also as frameworks/tools. On the other hand, a particular set of metrics does not necessarily reflect all aspects of quality. Obviously, further research in this area is required to validate the findings.

6. CONCLUSIONS

In this paper we presented the results of a case study aiming at the analysis of the evolution of software libraries. The motivation stems from the general belief that libraries excel in terms of their design quality. The analysis consisted in the examination of whether a trend is present in the evolution of three well-known object-oriented metrics for twenty open-source libraries. Although a definite conclusion could not be reached on whether an overall trend exists, the findings clearly revealed that libraries either remain stable or gradually improve in terms of quality.

ACKNOWLEDGMENTS

This research work is co-founded by the European Social Fund and National Resources, ESPA 2007-2013, EDULLL, "Archimedes III" program.

7. REFERENCES

[1] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "The financial aspect of managing technical debt: A systematic literature review," *Inf. Softw. Technol.*, vol. 64, pp. 52–73, Aug. 2015.

[2] V. Basili, G. Caldiera, and H. D. Rombach, "Goal Question Metric (GQM) Approach," in *Encyclopedia of Software Engineering*, John Wiley & Sons, Inc., 2002.

[3] J. Bevan, E. J. Whitehead,Jr., S. Kim, and M. Godfrey, "Facilitating Software Evolution Research with Kenyon," in *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA, 2005, pp. 177–186.

[4] J. Bloch, "How to Design a Good API and Why It Matters," in *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*, New York, NY, USA, 2006, pp. 506–507.

[5] T. Chaikalis and A. Chatzigeorgiou, "Forecasting Java Software Evolution Trends employing Network Models," *IEEE Trans. Softw. Eng.*, vol. PP, no. 99, pp. 1–1, 2015.

[6] T. Chaikalis, G. Melas, E. Ligu, and A. Chatzigeorgiou, "Seagle: Effortless Software Evolution Analysis," presented at the 30th International Conference on Software Maintenance and Evolution (ICSME'2014), Victoria, British Columbia, Canada, 2014, pp. 581–584.

[7] A. Chatzigeorgiou and E. Stiakakis, "Benchmarking library and application software with Data Envelopment Analysis," *Softw. Qual. J.*, vol. 19, no. 3, pp. 553–578, Sep. 2011.

[8] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994.

[9] D. Dig and R. Johnson, "How do APIs evolve? A story of refactoring," *J. Softw. Maint. Evol. Res. Pract.*, vol. 18, no. 2, pp. 83–107, Mar. 2006.

[10] M. W. Godfrey and D. M. German, "The past, present, and future of software evolution," in *Frontiers of Software Maintenance, 2008. FoSM 2008.*, 2008, pp. 129–138.

[11] R. Jabangwe, J. Börstler, D. Šmite, and C. Wohlin, "Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review," *Empir. Softw. Eng.*, pp. 1–54, Mar. 2014.

[12] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proc. IEEE*, vol. 68, no. 9, pp. 1060–1076, Sep. 1980.

[13] T. McDonnell, B. Ray, and M. Kim, "An Empirical Study of API Stability and Adoption in the Android Ecosystem," in *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, Washington, DC, USA, 2013, pp. 70–79.

[14] T. Mens and S. Demeyer, *Software Evolution*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[15] D. L. Parnas, "Software Aging," in *Proceedings of the 16th International Conference on Software Engineering*, Los Alamitos, CA, USA, 1994, pp. 279–287.

[16] S. Raemaekers, A. van Deursen, and J. Visser, "Measuring software library stability through historical version analysis," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, 2012, pp. 378–387.

[17] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*, 1 edition. Hoboken, N.J: Wiley, 2012.

[18] J. Tulach, *Practical API design: confessions of a Java framework architect*. [New York]; New York: Apress ; Distributed to the book trade worldwide by Springer Science+Business Media New York, 2012.

[19] "Trends in Software Libraries," *Trends in Software Libraries*, 30-Mar-2015. [Online]. Available: http://se.uom.gr/index.php/trends-in-software-libraries/. [Accessed: 30-Mar-2015].

[20] "R: The R Project for Statistical Computing," *The R Project for Statistical Computing*. [Online]. Available: www.r-project.org. [Accessed: 27-Mar-2015].