Exploring the Relationship between Software Modularity and Technical Debt

Peggy Skiada, Apostolos Ampatzoglou, Elvira-Maria Arvanitou, Alexander Chatzigeorgiou, Ioannis Stamelos

Department of Informatics, Aristotle University, Thessaloniki, Greece

Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece

peggy.skiada@gmail.com, apostolos.ampatzoglou@gmail.com, earvanitoy@gmail.com, achat@uom.gr, stamelos@csd.auth.gr

Abstract—**Modularity is one of the key principles of software design. In order for a software system to be modular, it should be organized into modules that are highly coherent internally, whereas at the same time as independent from other modules as possible. In this paper we explore coupling and cohesion metrics at the software package level—i.e., one of most basic levels of software functional decomposition in object-oriented (OO) systems, with the aim of investigating their relation to the technical debt of each package. Current state-of-the-art tools in TD measurement are working on the source code level, and the extent to which they can unveil limitations at the architecture level (e.g., violations of the modularity principle), has not been explored so far. To achieve this goal, we conducted a case study on 1,200 packages retrieved from 20 well-known open source software projects. The results of the study suggested that current measures of technical debt are able to identify / predict modules that lack modularity, and therefore suffer from Architectural Technical Debt (ATD). The results of the study are discussed both from the practitioners' and researchers' point of view.**

Keywords—coupling; cohesion; modularity; technical debt

I. Introduction

Technical Debt (TD) spans across all phases of the software development lifecycle, including requirements engineering, design, implementation and testing. [10]. In the general context, technical debt refers to immature work in a software system that takes compromises in one dimension to meet urgent needs in some other dimension [3]. In this work, we focus on technical debt at architecture level [12], i.e., architectural technical debt (ATD). ATD is caused by design decisions that consciously or unconsciously compromise system-wide quality attributes (QAs), especially maintainability [10] in order to speed up product delivery. Typically, ATD includes violations of best architecture practices and principles (e.g., lack of modularity), or the consistency and integrity of the software architectures (e.g., breaking a layered architecture).

According to van Vliet [18], high-level software design should be guided by four main principles, namely: provide abstraction, impose modularity, enforce information hiding, and decrease complexity. Among those, in this paper we focus on software modularity. According to ISO/IEC 25010 standard [2], modularity is one of the sub-characteristics of maintainability, which is one of the QAs compromised by ATD. Modularity is defined as the "*degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components* [2]".

To assess modularity, two quality properties need to be quantified, i.e., coupling and cohesion. Improved modularity is achieved by promoting low coupling and high cohesion. Coupling represents the strength of the connection between modules [18], whereas cohesion the "glue" that keeps a module together [18]—see Section III. In a typical OO system, classes (i.e., the most central element in OO) are grouped together in packages, based on their functional similarity. In order for an object-oriented design to be modular, classes of the same package are expected to highly interact with each other (high-cohesion, such as the classes within packages A and B in Figure 1), whereas dependencies among classes belonging to different packages should be limited [18] (low-coupling, such as the single dependency of A1 upon B1 in Figure 1).
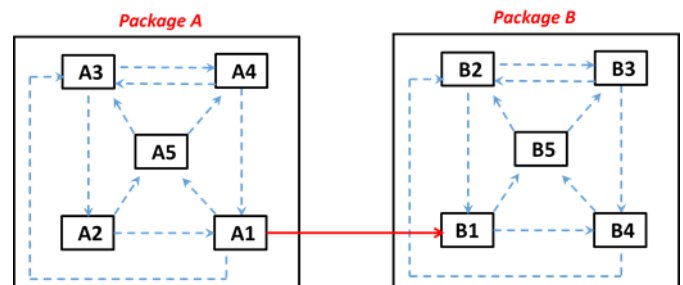

Fig. 1. Modularity Example

In this study, we investigate the relation between package-level modularity metrics and technical debt principal, as measured by a state-of-the-art tool, namely SonarQube that is based on the SQALE model [11]. SonarQube is an Open Source Software (OSS) platform for the continuous inspection of code quality. SonarQube assesses the technical debt of a software system at the source code level by counting violations of best practices (e.g., *Removal of Unused Private Fields*, *Constructors Should be Used for String Initialization*, or *@Override Should be Used for Overriding Methods*) and estimating the time needed to resolve them. The platform supports a plethora of programming languages and it can offer detailed reports regarding duplicated code, coding standards, unit tests etc. Nevertheless, based on the current TD calculation model it is not clear if the violations that SonarQube can capture are also (directly or indirectly) related to architecture violations, or at least, if the provided measure is correlated with violation of some architecture best practices (e.g. modularity). Although, at a first glance, the SonarQube TD calculation seems irrelevant to software package modularity, we suppose the existence of an un-

derlying relation, in the sense that sloppy implementation (as suggested by the presence of low-level code violations), might reflect flaws in the architectural design as well. The existence of such a relation would be helpful to both researchers and practitioners, since an easy to calculate index at the source code level (i.e., SonarQube TD), which is always available to software development teams, can act as a proxy of a more high-level and abstract concept (i.e., architectural modularity). To achieve this goal we have performed a large-scale empirical study on more than 1,000 packages written in Java and explored the aforementioned relationship.

The rest of the paper is organized as follows: In Section II we present a narrow (due to space limitations), but representative, related work on ATD measurement. In Section III, we outline the case study design, and in Section IV we present the results. We conclude the paper with a discussion of the findings and threats to validity in Section V.

II. RELATED WORK

Marinescu proposed an approach to identify and measure technical debt of object-oriented software systems by detecting and assessing specific types of design flaws through object-oriented metrics [13]. The approach is composed of four steps: (1) choose a set of concerned design flaws, (2) define rules for detecting the selected design flaws, (3) measure the negative impact of each instance of the design flaws, and, finally, (4) calculate an overall score based on all detected design flaws to indicate the design quality of a system. The accuracy of the technical debt measurement in this approach depends on the ability of the design flaws detection. This approach can only identify and measure technical debt at detailed design level, while our investigation focuses on ATD.

Nord et al. defined a metric for managing ATD [15]. The value of this metric, calculated for each release, is the total cost of the implementation of new architectural elements introduced in this release, and the rework of pre-existing elements in previous releases. They considered architectural rework as the necessary adaption work for adding new architectural elements to the existing architecture of a software system. The rework cost is calculated based on the analysis of the changing dependencies from existing adapted architectural elements to the new introduced elements. This metric can be used to calculate the relative amount of ATD incurred in different software evolution paths, i.e., release plans. Suppose that there are two release plans RP1 and RP2, in which the same features are implemented, i.e., they generate the same amount of business value. The relative amount of ATD is the difference between the values of metric calculated on RP1 and RP2. This metric can facilitate architecture decision-making. The main limitation of this approach is the accuracy of the estimation of implementing new features and rework, especially the latter. Each software evolution path involves several releases, which implies that the estimation of rework and new implementations of later releases is based on the estimation of the earlier releases. This may pose a significant threat to the accuracy of ATD estimation.

III. CASE STUDY DESIGN

Research Goals and Research Questions. The main research question of this paper is: "*Is TD principal as quantified by So-* *narQube related to the lack of software modularity?*" The answer to this question can unveil if and to what extent SonarQube is able to capture ATD, in the sense that low modularity is an architecture best practice violation. To answer this main question we pose two more specific questions, based upon the two quality properties that compose modularity: i.e., coupling and cohesion.

   RQ$_1$: *Is TD principal related to package cohesion?*
   RQ$_2$: *Is TD principal related to package coupling?*
The study has been designed and reported according to the template suggested by Runeson et al. [16].

Case Selection and Units of Analysis. This study is a holistic multiple case study in which cases and units of analysis are software packages. As subjects for this study, we used 20 Java OSS projects, which have been selected based on the following criteria—for more details see Arvanitou et al. [6]:

- ***The software is a popular OSS project*** in Sourceforge.net. This criterion ensures that the investigated projects are recognized as important by the OSS community.
- ***The software has more than 20 versions*** (official releases). We have included this criterion for similar reasons to c1.
- ***The software contains more than 300 classes***. This criterion ensures that we will not include "toy examples" in our dataset.
- ***The software is written in java***. We include this criterion because the employed metric calculation tools analyse Java bytecode.

To measure TD (more specifically its principal) we used version 6.3 of SonarQube, without further configuration and according to its default status. TD principal at the package level is automatically calculated by SonarQube as the sum of the TD principal of all classes in the package. Coupling and cohesion at the package level have been quantified using three metrics:

- ***ACa – Average Coupling Afferent*** This metric represents the average afferent coupling of packages. Afferent coupling is the number of outgoing dependencies of a package to other packages [14].
- ***TCIP – Total Coupling Intensity between Packages***: This metric represents the count of class dependencies that span among different packages. This metric is inspired by the traditional Coupling between Objects metrics [8], which is calculated at the class level. The idea of employing two coupling metrics is that one (ACa) captures the number of dependencies at the architecture level, whereas the other (TCIP) the intensity of the dependency [4].
- ***CaPC - Cohesion among Package Classes***: This metric assesses how closely two classes that belong to the same package collaborate with each other. The metric is inspired by reversing the calculation of Lack of Cohesion of Methods [8]. To calculate this metrics we compute the total number of pairs of classes that belong to one package, and then we investigate the percentage of these pairs that are coherent (i.e., they are coupled to each other).

The tool that we used for calculating these metrics has been developed in our groups as part of a series of previous studies on change impact analysis [4][5][6]. In the end of this process the dataset of this study consisted of approximately 1,200 software packages.

Data Collection and Analysis. Each package of our dataset (i.e., row) is characterized by five variables: *name*, *TD Principal*, *ACa*, TCIP, and *CaPC*. To answer the aforementioned research questions we perform: (a) Spearman and Pearson Correlation between *ACa*, TCIP and TD, and *CaPC* and TD Principal, (b) Univariate Regression Analysis with dependent variables: *ACa*, TCIP and *CaPC*, and (c) hypothesis testing to check the discriminative power of TD to identify classes with low cohesion and high coupling. The analysis follows the IEEE standards' guidelines for empirical metric validation [1].

IV. RESULTS

In this section we present the results that have been obtained from data analysis. In Table I we present the results for correlation analysis (Spearman and Pearson), and predictive power. For presenting the results on *Correlation* and *Consistency*, we use the correlation coefficients (coeff.) and the levels of statistical significance (sig.). The value of the coefficient denotes the degree to which the value of the modularity metric is in analogy to the value of TD (either as an actual value or as a ranking, respectively). For reporting on *Predictive Power*, with a regression model, we present the level of statistical significance of the effect (sig.) of the independent variable on the dependent (how important is the predictor in the model), and the accuracy of the model (i.e., mean standard error). The results of Table I suggest that **TD principal is value-wise mostly correlated to TCIP, ranking-wise to CaPC, whereas it most accurately predicts the value of CaPC**.

TABLE I. CORRELATION ANALYSIS AND PREDICTIVE POWER

Assessment Criterion	Test	Indicator	ACa	TCIP	CaPC
Correlation	Pearson Correlation	Correl. Coeff.	0.102	0.372	0.314
		Sig.	0.01	0.00	0.00
Consistency	Spearman Correlation	Correl. Coeff.	0.318	0.493	0.513
		Sig.	0.00	0.00	0.00
Predictive Power	Linear Regression	Std. Error	0.332	0.716	0.310
		Sig.	0.01	0.00	0.00

Additionally, for exploring the *Discriminative Power* of TD principal, we investigate whether groups of packages differ with respect to the corresponding modularity metric score. The groups of packages have been created using the equal frequency binning technique [19]. For reporting on the hypothesis testing, we present the level of statistical significance (sig.) and the F-value of the ANOVA. We note that in order for TD to adequately discriminate groups of cases, the significance value should be less than 0.05, or 0.01 for strict evaluations. In the case of our study, we preferred to use the 0.01 threshold since many differences were significant at the 0.05 level, leading to inconclusive results. The results are visualized by 95% confidence interval (CI) bars. The 95% CI Bars present the mean value of a numerical variable and its 95% confidence interval. Error bars can be used to visually compare mean values of two or more groups and get preliminary indications on the existence of significant differences. As suggest-

ed by Fig. 2 and ANOVA, **TD principal is capable of discriminating the various levels of all metrics: Optimal discrimination is achieved for CaPC** (F: 41.775, sig < 0.01).
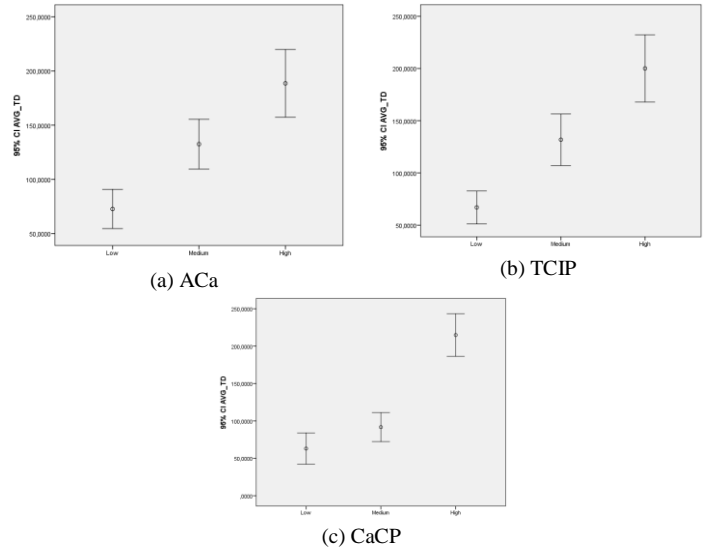

(a) ACa

(b) TCIP

(c) CaCP

Fig. 2. Discriminative Power

Finally, another interesting observation is that a combined metric based on TCIP and CaPC (i.e., a division of cohesion by coupling [17]) is capable of achieving top results in Correlation analysis (coef: 0.408), acceptable results for Consistency (coef: 0.495), and by far the highest predictive power (R^2: 91.9%).

V. DISCUSSION

Interpretation of results. The main finding of this case study is that *TD principal is able to assess the levels of software modularity at the package level*. A possible interpretation of this relation is that software quality assurance processes usually span across multiple development phases: thus, if a development team is not interested in removing (or not introducing) violations of source code best practices, it is highly likely that similar problems have been neglected in previous development phases as well (i.e., architecture). For example, one could assume that in case a development team is interested in reducing the coupling of classes or the number of 'static' imports, both of which are classified as brain-overload smells by SonarQube, then the architectural quality of the system has also been a concern for the team. More specifically, TD is moderately correlated to coupling and cohesion (as actual value), and strongly correlated to these metrics, by considering rankings. The fact that rank correlation achieves better results than value correlations is expected due to the difference of the range of values of the involved variables. Additionally, TD principal is able to predict the values of coupling and cohesion metrics at a statistically significant level, with however low accuracy. Nevertheless, it should be noted that TD is able to accurately (approx. 90%) predict the value of cohesion divided by coupling. This finding is particularity interesting in the sense that the TD assessment offered by SonarQube can be useful at both the implementation and the architecture design level.

By *comparing coupling and cohesion metrics* that we used, one can observe that: (a) TD principal seems to be better related to cohesion, rather than coupling; and (b) TD principal is more closely related to TCIP rather than ACa. A main difference between the three metrics is that CaPC is a bounded metric (expressed as percentage), which therefore can be more easily interpreted. This finding is in accordance to previous studies suggested that bounded metrics are more strongly correlated with the existence of bad smells [9]. Regarding the two coupling metrics, the intensity of the dependencies seems to be more relevant to high level qualities, compared to a simple count of dependencies. This outcome is affirmed in previous studies (e.g., the MPC metric performs better than CBO) [5].

Implications to researchers & practitioners. The outcomes of this study provide some useful implications to researchers and practitioners. First, it is suggested (although further validation is required) that the SQUALE Index as calculated by SonarQube, is able to assess a specific aspect of Architectural Technical Debt (i.e., modularity). This ability of SonarQube TD is considered helpful in the sense that ATD is an abstract concept, which can be quantified through artifacts that are rarely present in practice (e.g., decision document, component diagrams, etc.), and therefore can mostly be estimated through proxies. On the other hand, TD calculation with SonarQube is an easier and straightforward process which relies only on the existence of source code, which is always available. Therefore, we encourage software architects to take into account package level TD assessments offered by SonarQube as a proxy of package modularity. Furthermore, we encourage researchers to investigate other possible underlying relations between source code TD and ATD. Although the source code might have been substantially drifted from the intended architecture, and the decisions made at that stage, however it represents the implemented architecture, and the way that the aforementioned decisions have been put into practice.

Threats to validity. Due to space limitations, only the major threats to validity of this study are presented in this section. Regarding the study constructs, although the relation between modularity and coupling / cohesion is not disputable, the investigated metrics have not been validated prior to their use. However, by taking into account that all metrics stem from a straightforward tailoring of well-known and rigorously validated metrics at the class level, to some extent ensures their validity. The success of such tailoring approaches has been discussed and empirically validated in previous work [4][9].

ACKNOWLEDGMENT

REFERENCES

[1] 1061-1998: IEEE Standard for a Software Quality Metrics Methodology, *IEEE Standards*, IEEE Computer Society, 31 December 1998 (reaffirmed 9 December 2009).

[2] 25010-2011 ISO/IEC Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — *System and software quality models*. pp. 1-34, 2011.

[3] Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., Maccormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K., and Zazworka, N., "Managing technical debt in software-reliant systems", *Workshop on Future of software engineering research (FoSER'10)*, ACM, New Mexico, USA, pp. 47-52, 2010.

[4] E. M. Arvanitou, A. Ampatzoglou, K. Tzouvalidis, A. Chatzigeorgiou, P. Avgeriou, and I. Deligiannis, "Assessing Change Proneness at the Architecture Level: An Empirical Validation", *International Workshop on Emerging Trends in Software Design and Architecture (WETSoDA 2017)*, IEEE, 2017.

[5] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "Introducing a ripple effect measure: a theoretical and empirical validation", *9th Int. Symposium on Empirical Software Engineering and Measurement (ESEM '15)*, IEEE, 22–23 October 2015, China.

[6] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou. "Software Metrics Fluctuation: A Property for Assisting the Metric Selection Process", *Information and Software Technology*, 72 (4), 2016.

[7] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "A Method for Assessing Class Change Proneness", *21st International Conference on Evaluation and Assessment in Software Engineering (EASE' 17)*, ACM, 15-16 June 2017, Sweden.

[8] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design", *Transactions on Software Engineering*, IEEE Computer Society, 20 (6), pp. 476 - 493, June 1994.

[9] S. Charalampidou, A.Ampatzoglou, and P. Avgeriou., "Size and cohesion metrics as indicators of the long method bad smell: An empirical study", *11th Intern. Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE '15)*. ACM, 2015.

[10] Kruchten, P., Nord, R.L., and Ozkaya, I., "Technical Debt: From Metaphor to Theory and Practice", *IEEE Software*, 29 (6), 18-21, 2006.

[11] J. L. Letouzey and T. Coq, "The SQALE Analysis Model: An Analysis Model Compliant with the Representation Condition for Assessing the Quality of Software Source Code", *2nd International Conference on Advances in System Testing and Validation Lifecycle (VALID' 10)*, IEEE Computer Society, pp. 43-48, Nice, Paris, 22-27 August 2010.

[12] Z. Li, P. Avgeriou, and P. Liang. "A Systematic Mapping Study on Technical Debt and Its Management", *Journal of Systems and Software*, 101(3):193–220, 2015.

[13] R. Marinescu, Assessing technical debt by identifying design flaws in software systems. *IBM J of Res. and Develop.*, 56 (5), pp. 1-13, 2012.

[14] R. C. Martin "Agile software development: principles, patterns and practices", *Prentice Hall*, New Jersey. 2003.

[15] Nord, R.L., Ozkaya, I., Kruchten, P., and Gonzalez-Rojas, M., "In search of a metric for managing architectural technical debt", *10th Working IEEE/IFIP Conference on Software Architecture (WICSA '12)*, IEEE, Helsinki, Finland, 2012.

[16] Runeson P., Höst M., Rainer A., and Regnell B., "Case Study Research in Software Engineering: Guidelines and Examples", *John Wiley and Sons, Inc*, 2012.

[17] N. Tsantalis and A. Chatzigeorgiou, "Identification of Move Method Refactoring Opportunities," *IEEE Trans. Softw. Eng.*, 35 (3), May 2009.

[18] H. van Vliet, "Software Engineering: Principles and Practice", *John Wiley & Sons*, 2008.

[19] I. Witten and E. Frank, "Data Mining: Practical machine learning tools and techniques", *Morgan Kaufmann*, 2nd Edition, 2005