Structural Quality Metrics as Indicators of the Long Method Bad Smell: An Empirical Study

Sofia Charalampidou, Elvira-Maria Arvanitou, Apostolos Ampatzoglou, Paris Avgeriou,
Alexander Chatzigeorgiou, Ioannis Stamelos

Department of Computer Science, University of Groningen, Netherlands
Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
Department of Informatics, Aristotle University, Thessaloniki, Greece
s.charalampidou@rug.nl, earvanitoy@gmail.com, apostolos.ampatzoglou@gmail.com, paris@cs.rug.nl, achat@uom.gr, stamelos@csd.auth.gr

***Abstract*—Empirical evidence has pointed out that Extract Method refactorings are among the most commonly applied refactorings by software developers. The identification of Long Method code smells and the ranking of the associated refactoring opportunities is largely based on the use of metrics, primarily with measures of cohesion, size and coupling. Despite the relevance of these properties to the presence of large, complex and non-cohesive pieces of code, the empirical validation of these metrics has exhibited relatively low accuracy (max precision: 66%) regarding their predictive power for long methods or extract method opportunities. In this work we perform an empirical validation of the ability of cohesion, coupling and size metrics to predict the existence and the intensity of long method occurrences. According to the statistical analysis, the existence and the intensity of the Long Method smell can be effectively predicted by two size (LoC and NoLV), two coupling (MPC and RFC), and four cohesion (LCOM1, LCOM2, Coh, and CC) metrics. Furthermore, the integration of these metrics into a multiple logistic regression model can predict whether a method should be refactored with a precision of 89% and a recall of 91%. The model yields suggestions whose ranking is strongly correlated to the ranking based on the effect of the corresponding refactorings on source code (correl. coef. 0.520). The results are discussed by providing interpretations and implications for research and practice.**

***Keywords*—** long method; coupling; cohesion; size; case study

I. Introduction

Code smells are warning signs indicating possible deeper problems in the design or code of software, often resulting from the violation of at least one programming principle [9]. The notion of smells has been widely adopted by software developers because such problems are usually easy to identify and perceive and in most of the cases can be resolved by an appropriate refactoring that eliminates the smell without modifying system behaviour [11]. A number of methods and tools have been proposed to assist in the identification of code smells and the automatic application of refactorings. Smell detection approaches have evolved over the years yielding very sophisticated tools for some of the code and design problems. The first approaches relied on structural information or software metrics [19], but over the years researchers have acknowledged the need to resort to textual information as well [21]. Change history information has also been used to improve the accuracy of smell detection [22]. Feedback-based smell detection approaches have been introduced to adapt the employed thresholds [16] and even learning-based techniques that train probabilistic models on golden sets have tackled the problem of identifying meaningful refactoring opportunities. However, the use of metrics remains a universal and generalizable approach to identify parts of the code base in need of refactoring, and for characterizing the intensity of the underlying problems.

In this study we propose such a metric-based approach, focusing on one specific smell – the *Long Method*, which is resolved through the *Extract Method* refactoring, as defined by Fowler et al. [9]. The reasons for working with this smell are the following. First, ***long method is a frequently occurring smell*** [5]. Chatzigeorgiou and Manakos [5] investigated through a case study, the presence and evolution of four types of code smells, i.e., Long Method, Feature Envy, State Checking and God Class. The results indicated that long methods were considerably more common compared to the other smells. Second, ***long methods are of particular urgency as they often occur in the early versions of software and persist unless targeted refactoring activities are performed***. Specifically, a case study on an open source project (jFlex) revealed that 89.8% of the long methods identified in that project remained unresolved in all the explored versions [5]. Furthermore, Palomba et al. [21] have found that Long Method exhibited the largest diffuseness among the studies code smells (it affects 84% of the analysed releases) and on average each of the examined releases is affected by 44 Long Method instances. Additionally, so far ***only a few metrics have been assessed with respect to their capacity to predict the existence of long methods***. To the best of our knowledge, current approaches achieve a recall rate of 59% and a precision rate between 39%-66% (see Section II). Finally, the ability of metrics to prioritize the intensity of the long methods to be resolved has not been empirically investigated yet. Metric-based detection approaches are based on the fact that the presence of a smell in a method or class negatively impacts certain quality properties [2] (e.g. a 'God' class has low maintainability). The affected properties can then be diagnosed by the use of the associated software metrics (e.g. size, cohesion and complexity can be used to detect a 'God' class).

Based on a previous study, size and cohesion metrics are closely related to the existence of long methods [4]). Additionally, we study the coupling property, since coupling and cohe-

sion are two closely connected qualities (see [25]). Therefore, we focus on three quality properties: *method size, coupling,* and *cohesion*: In particular, we empirically investigate the ability of size, coupling and cohesion metrics: (a) to *predict which methods suffer from the long method bad smell*, and (b) to *prioritize the intensity of the smell*, based on the extract method opportunities that they present. Additionally, we will investigate whether the *combination of the studied metrics could provide us a prediction model* with higher precision and recall rates regarding the concerns stated in (a) and (b). We note that we assess the intensity of the problems caused by a long method, based on the identified extract method opportunities, since Fowler et al. [9] suggest that in 99% of the cases, the extract method refactoring (i.e. removal of code chunks from one method that can be turned into new methods, whose names explain their purpose) is the solution to the long method bad smell. This paper is a follow-up version of a case study that aimed at the assessment of one size and several cohesion metrics as indicators of the existence of long methods, and their intensity [4]. The main points of differentiation compared to [4] are: (a) the assessment of one additional quality attribute, i.e., coupling, by investigating two coupling metrics; (b) the assessment of three additional size metrics (compared to only one); and (c) the use of multiple regression to investigate the joint predictive power of size, cohesion, and coupling metrics.

II. Related Work

As related work for this study we consider papers that evaluate existing metrics, with respect to their ability of detecting the long method smell. Thus, studies which use other approaches for identifying software artifacts that suffer from bad smells (e.g., [24]), have been excluded from this section. Also, we omitted papers that propose new metrics for identifying refactoring opportunities or bad smells (e.g. [23]). An extensive analysis of related work is provided in [4]. Our contributions compared to state-of-the-art can be summarized as follows:

- the investigation of the predictive power of metrics for the identification of long method smells is extended to cohesion, coupling, and size metrics.
- Cohesion, coupling, and size metrics are compared not only as predictors of the existence of long methods but also as predictors of the intensity of the design problem.
- The proposed model provides higher accuracy (precision and recall), compared to existing detection approaches.

III. Metrics Selection

According to Fowler et al. [9], a method is characterized as long, based on: (a) the functional distance of the lines of code of its body, and (b) its size. The functional distance is related to ***cohesion***, which is defined as the functional relatedness of the elements of a module [17]. Method-level (i.e., inside the method body) cohesion metrics have been firstly discussed by Yoshida et al. [26], when introducing the NCCP metric, which has been derived from the transformation of the SCOM metric [8]. In our approach, we have applied a process, similar to the one proposed by Yoshida et al. [26], for all 13 low-level cohesion metrics collected by Al Dallal [1]. The main principles for this process are the mapping of: (a) Lines of code to methods, and (b) all variables within the scope of the method (i.e., attributes, local variables, or parameters) to attributes [4]. For example, the classic LCOM2 metrics [6] is calculated as follows. In total, in this study we used 13 cohesion metrics [4].

LCOM2 = P – Q, if P − Q ≥ 0 / otherwise LCOM2 = 0,

where P is the number of pairs of lines that do not share variables, and Q is the number of pairs of lines that share variables.

Additionally, we consider ***coupling*** metrics (MPC and RFC) because of the relationship between cohesion and coupling. Both metrics are explicitly related to method calls, because at method level, methods are interconnected only via method invocations. The tailoring of the two metrics is as follows:

- *Message Passing Coupling (MPC)* [14] is originally defined at class level as the sum of the number of method calls made by all members of a class. For the needs of this study we tailor this definition by counting the total number of method invocations inside a single method.
- *Response for a Class (RFC)* [6] is also defined at class level as the count of (public) methods in a class and methods directly called by them. Based on the goal of this study we need to tailor this definition, so as to calculate RFC at the method level. Thus we define the RFC metric at method level as the number of "local methods" (i.e. the studied method itself), plus the count of unique method invocations inside its body. Therefore, its difference with MPC lies on the fact that RFC counts multiple invocations of the same method as one, whereas MPC reports also on how many times a method is called. The differences between RFC and MPC are discussed by Li and Henry [14].

Finally, concerning method size, we have used four metrics, one related to the size of methods in terms of statements, and three related to the availability of variables in the scope of the method. The number of available variables is indirectly related to cohesion, in the sense that in order for a method to be cohesive, its whole body should work on the same variables. Thus, the more variables are accessible, the more probable the method to be non-coherent. The following metrics have been used:

- *Lines of Code (LoC)* [10]. Total number of statements inside the method body, excluding blank lines and comments.
- *Number of Attributes (NoA)* [10]. Total number of attributes in the class that the studied method belongs to.
- *Number of Method Parameters (NoMP)* [10]. Total number of method parameters.
- *Number of Local Variables (NoLV)* [13]. Total number of local variables in the method.

IV. Case Study Design

Objectives and Research Questions. Our goal is to *analyse* 19 structural quality metrics and their combinations *for the purpose of* evaluation, *with respect to* their ability to: (a) predict the existence of long method, and (b) predict their intensity, *from the viewpoint of* software engineers, *in the context of* java open source software", which decompose to three questions.

RQ1: *Which cohesion, coupling, and size metrics can be used for predicting the existence of long methods?*

RQ2: *Which cohesion, coupling, and size metrics can be used for ranking long methods, with respect to their intensity?*

RQ3: *Can the refactoring prediction and prioritization be improved by combining structural metrics?*

Case Selection and Units of Analysis. The study design is a replication of previous work (i.e., [4]), which focused only on a subset of the metrics that we handle in this paper (cohesion ones). The study is a holistic multiple case study: methods are both the cases and the units of analysis.

Data Collection and Pre-Processing. For every method, we recorded the following variables:

- **V1 – V3:** Project, class, and method name. This set of variables is used only for characterization purposes.
- **V4 – V16:** Twelve cohesion metrics: Independent variables to be analysed.
- **V17-V20:** Four size metrics (see metrics described in Section III): Independent variables to be analysed.
- **V21 – V22:** Two coupling metrics (see metrics described in Section III): Independent variables to be analysed.
- **V23:** Long Method (yes / no). This variable was assigned a binary score from the developer of each project. This variable is going to be used as dependent variable in RQ_1.
- **V24:** Long Method Intensity. This variable corresponds to the average number of lines to be extracted if the extract method refactoring is applied. This variable is going to be used as dependent variable in RQ_2.

As pre-processing, we filtered out of the dataset methods that were less probable to suffer from the long method smell. The motivation for this was to build a balanced dataset with respect to the number of methods that are in need of refactoring and those that are not. According to King and Zeng [12], applying predictive models (e.g. regression) in rare events datasets (in our case 10%), can benefit from case selection strategies that reduce the number of negative events (in our cases methods that are not in need for refactoring). To this end, we excluded from our dataset methods, which are smaller than 30 lines of code: according to Lippert and Roock [15], a method is prone to suffer from bad smells if its size exceeds 30 lines of code. After applying this filter, the dataset was comprised of 79 methods (including 40.5% of negative events). Although the number of cases seems rather small, the number of cases is limited due to the involvement of human experts and the manual processing of source code.

Data Analysis. To answer the research questions, we have statistically analyse data, as presented in Table I.

TABLE I. DATA ANALYSIS OVERVIEW

	Variables	Statistical Analysis
RQ_1	Cohesion, Coupling, and Size metrics Long method (yes / no)	Simple Logistic Regression
RQ_2	Cohesion, Coupling, and Size metrics Extract Method Intensity	Spearman Correlation
RQ_3	Cohesion, Coupling, and Size metrics Long method (yes / no) Extract Method Intensity	Multiple Logistic Regression Multiple Linear Regression Spearman Correlation

V. RESULTS

Due to the size of our results, we preferred to present only the statistically significant ones. Interpretations and implications to researchers and practitioners are provided in Section VI.

Metrics for predicting the existence of long methods. To assess the ability of metrics to predict whether a method suffers from the long method smell (RQ_1), we present the results of the corresponding regression analysis (see Table II): We present two sets of measures: the first is related to the construction of the prediction model (i.e., beta values and significance), whereas the second is related to its evaluation (accuracy, precision, and recall). In the table we present only metrics that have a predictive power at a statistically significant level (sig.≤ 5%). The results of Table II suggest that in total nine metrics are able to predict which methods are in need for refactoring. We observe that two size metrics (i.e. LoC and NoLV), three not normalized cohesion metrics (i.e., LCOM, LCOM2 and LCOM4), and one coupling metric (MPC), form a group of measures significant at the 1% level. Finally, we can observe that two normalized cohesion metrics (CC and Coh) are able to predict methods that are in need of extract method refactoring with a precision between 89-96%. As expected, these two metrics misclassify a larger number of false-negatives compared to the rest of the metrics, leading to a slightly decreased recall rate. The highest recall and accuracy are offered by the LoC metric, followed by LCOM1.

TABLE II. PREDICTIVE POWER OF METRICS FOR LONG METHOD

	Prediction Model			Predictive Power		
Metric	b0	b1	sig.	Accuracy	Precision	Recall
LoC	-4.491	0.103	0.00	84.8%	80.85%	92.68%
LCOM1	-1.281	0.002	0.00	79.7%	74.47%	89.74%
LCOM2	-0.809	0.002	0.00	70.9%	68.09%	80.00%
MPC	-1.939	0.054	0.01	79.7%	84.4%	80.9%
LCOM4	-0.536	0.113	0.01	68.4%	76.60%	72.00%
Coh	1.392	-10.200	0.03	62.0%	89.36%	62.69%
RFC	-1.846	0.154	0.04	73.4%	75.0%	82.9%
NoLV	-1.063	0.122	0.04	69.6%	71.7%	80.9%
CC	0.996	-5.362	0.05	68.4%	95.74%	66.18%

Metrics for long method prioritization. To assess the ability of metrics to rank methods, based on their intensity (RQ_2), we summarize the results of the Spearman correlation test in Table III. Specifically, we present the ability of the examined metrics to rank methods, based on the average number of lines that will be extracted, if a proposed refactoring is applied. We note that the sign of the correlation depends on whether the metric expresses cohesion (e.g., Coh), or lack of cohesion (e.g., LCOM1). Concerning size and coupling metrics, the sign is positive, due to their direct relation to the number of extract method opportunities. In Table III, we present only metrics that are significantly correlated to the dependent variable: (sign < 0.05 and r > 0.2)[1].

TABLE III. CORRELATION TO AVERAGE LINES TO BE EXTRACTED

Metric	Correl. Coefficient	Sig.	Metric	Correl. Coefficient	Sig.
LCOM1	0.472	0.00	Coh	-0.303	0.01
LoC	0.463	0.00	RFC	0.292	0.01
LCOM2	0.385	0.00	CC	-0.268	0.02
MPC	0.366	0.00	DC_D	-0.254	0.02
NoLV	0.355	0.00	SCOM	-0.253	0.03
LSCC	-0.326	0.00	LCOM5	0.244	0.03

Table III suggests that the results are similar to those of RQ_1, in the sense that LCOM1, LCOM2, LoC and MPC are the top ranked indicators of the smell intensity, followed by NoLV, Coh, RFC, and CC, which are also highly ranked in

[1] Correlations with r < 0.2 present weak or non-existing relations [18]

RQ1. An additional finding from comparing the results of RQ1 to those of RQ2 is the fact that LSCC, DCD, SCOM, and LCOM5 are valid indicators for the intensity of the smell, but not for the existence of extract method opportunities. On the other hand, LCOM4 is not able to rank smell intensity, despite the fact that it is a significant predictor of their existence.

Predictive power of combined metrics. To answer RQ_3 we performed a Multiple Stepwise Logistic Regression. The regression, started with the most influential variable, which according to the results was RFC.

TABLE IV. STRUCTURAL METRICS – LONG METHOD (PREDICTIVE POWER)

Step	Included Variable	Prediction Model		Predictive Power		
		Coeffi-cient	sig.	Accuracy	Precision	Recall
8	LoC	0.119	0.01	88.6%	89.4%	91.3%
	RFC	0.153	0.01			
	LCOM1	-0.00	0.02			
	NoMP	0.394	0.04			

The details of final step are reported in Table IV, to demonstrate the changes in predictive power. We note that for all steps the model was statistically significant (sig= 0.00). The results suggest that the combination of the four metrics (see final step of the model) achieves precision that equals 89.4%, and recall that equals 91.3% (accuracy: 88.6%). These results are considerably increased compared to those of analysis considering single variables (see RQ1)—i.e., LoC: 80.9% precision and 92.7% recall (84.8% accuracy). Considering the variables involved in the model, it is interesting to mention that all quality attributes (i.e. cohesion—LCOM1, coupling—RFC, and size metrics—LoC and NoMP) were involved, confirming that all are related to the existence of the long methods. To investigate whether the combination of multiple variables in a single model would provide a better indicator for the smell intensity, we performed a backward linear regression, starting with all potentially good indicators in step 1 and removing the least influential ones in every step. The formula that best predicts the Long Method Intensity (LMI) is:

LMI = -8.223 - 0.004 * LCOM2 + 0.398 * LoC - 0.082 * MPC - 22.671 * CC

We note that although some metrics do not appear to have a statistically significant correlation with the dependent variable (i.e., Extract Method Intensity), the overall model is statistically significant at the 0.05 level (i.e., sig = 0.03). To be able to compare the predictive power of the newly developed Long Method Intensity formula (LMI), with the one of single metrics, we performed a Spearman rank correlation between LMI and the smell intensity variable. The results indicated a strong correlation (coefficient: 0.520 and sig: 0.00), which is higher than the stronger correlation of a single metric (0.472 for LCOM1). Therefore, similarly to RQ_{3a}, a clear improvement in the predictive power has been achieved. Finally, metrics from all quality attributes have been used for the construction of the formula, affirming their importance for smell intensity.

VI. DISCUSSION / CONCLUSIONS

Comparison to related work. Our metric-based approach presents the highest precision. In particular, the precision of the individual metrics, proposed in this study, ranges from 68% to 93%, whereas related work reports 50% precision of complexity [19] and 38-66% precision of size metrics [7]. The precision of LoC, based on our results is 81%. We note that calculating recall for [7] and [19] was not possible. Based on our results CC and Coh present the highest precision. A safe comparison of the aforementioned findings can only be accomplished by applying all the approaches on a common dataset. Finally, by combining metrics, we achieve predictive power that clearly advances the state-of-the-art, since we accomplish 89.4% precision and 91.3% recall. The fact that the results obtained by the combined approach are better than those of individual metrics is expected, since models that assess the joint effect of metrics can accommodate the synthesis of characteristics of all independent variables. The rest of this section is devoted on discussing individual metrics.

Interpretation of results. Based on the results of this study, we argue that coupling, size, and cohesion metrics should be used for the identification of extract method opportunities, and subsequently for the mining of long method bad smell instances. This is a rather intuitive result, in the sense that all of them have been already associated in the literature as an indicator of the number of distinct functionalities that the module offers, which can be extracted in a new method [9]. Comparing the ability of ***individual size, coupling, and cohesion metrics*** to indicate if a method is long, cannot lead to safe conclusions, in the sense that nine metrics seem to outperform the rest, without large differences among them.

Two of the top three *cohesion metrics* (i.e., LCOM1 and LCOM2) are correlated to size (LoC), in the sense that they are open-ended metrics, whose upper limit is calculated as the count of combinations by two for the number of lines of code. Regarding precision two normalized metrics (i.e., Coh and CC) are the optimal predictors. Therefore, if one is interested in capturing as many long methods as possible, one should prefer LoC or a non- normalized cohesion metric (Coh or CC), whereas if one is interested to get as fewer false positives as possible, then one should prefer normalized cohesion metrics (LCOM1 or LCOM2) or a coupling metric (MPC or RFC). Similarly to O' Cinneide et al. [20], we suggest that different aspects of cohesion that metrics quantify, lead to different predictive and ranking power. Other interesting findings are:

- Metrics that quantify lack of cohesion (i.e., LCOM1, LCOM2, and LCOM4), appear to have higher predictive power concerning long method identification. This result is especially interesting since LCOM1 and LCOM2 have until now received much criticism in the literature (e.g., [3]). However, our results suggest that the fact that they are open-ended and that they are related to size, provides them with a comparative advantage for the identification and characterization of long methods.
- Metrics that involve method invocations in their calculation (i.e., LCOM4, TCC, LCC, DCD, and DCI) appear to provide lower ranking power, than those that omit them. Thus, we assume that the semantic distance between two lines is not related to whether they call the same method, but only to whether they access common variables.
- Although Coh is calculated as a function of LCOM5, it provides better results, implying that the suggested normalization by Briand et al. [3] (which drops the assumption that each attribute is referenced by at least one method) is more fitting compared to the original one.

Finally, by comparing *coupling metrics*, we were not able to identify important differences, since both studied metrics perform similarly in both the prediction of the existence of long methods, and the prediction of the smell intensity. By comparing *size metrics*, we observe that the Lines of Code (LoC) is optimal predictor, followed by the Number of Local Variables (NoLV). Both results are intuitive in the sense that:

- LoC is a dominant characteristics of long methods, and
- NoLV is expected to be related to method-specific functionalities. In particular, NoLV performs better than other metrics since: (a) NoA is the same among all methods of a class (i.e., it cannot differentiate methods in the same class), and (b) the parameters passed in a method (NoMP) are probably be used throughout the whole method, in contrast to local variables which have a more focused scope.

Implications for researchers & practitioners. We suggest that practitioners can benefit from the use of the metrics during quality monitoring tasks, especially in cases that bad smell detection tool support is not available for their programming environments. Additionally, software engineers can use the results of RQ_2 and RQ_3 to prioritize manual method code inspections, with respect to refactoring identification, since these metrics are strongly correlated to the intensity of the smell. In addition, researchers can use these results to develop approaches for identifying extract method opportunities, based on method-level cohesion metrics or explore the potentially improved predictive and ranking power of approaches using multi-criteria methods like the analytic hierarchy process (AHP), or Bayesian networks. Furthermore, they can investigate the possibility of identifying thresholds, for the nine metrics presenting the highest predictive power, that when surpassed a method can be classified as in need for extract method refactoring. Finally, it would be interesting to investigate if method-level metrics can be used for developing feature identification algorithms. The inherent relation between lack of cohesion and the number of functionalities might lead to a promising way for exploring the field of feature extraction.

ACKNOWLEDGMENT

Work reported in this paper: (a) has received funding from the European Union Horizon 2020 research and innovation programme under grant agreement No. 780572 (project: SDK4ED); and (b) was financially supported by the action "Strengthening Human Resources Research Potential via Doctorate Research" of the Operational Program "Human Resources Development Program, Education and Lifelong Learning, 2014-2020", implemented from State Scholarship Foundation (IKY) and co-financed by the European Social Fund and the Greek public (National Strategic Reference Framework (NSRF) 2014–2020).

REFERENCES

[1] J. Al Dallal and L. Briand,"A Precise method-method interaction-based cohesion metric for object-oriented classes", *Transactions on Software Engineering and Methodology*, ACM, 23 (2), Article 8, 2012.

[2] J. Bansiya and C. G. Davies, "A hierarchical model for object-oriented design quality assessment", *Transactions on Software Engineering*, IEEE Computer Society, 28 (1), pp. 4-17, January 2002.

[3] L. C. Briand, J. Daly, and J. Wuest, "A unified framework for cohesion measurement in object-oriented systems", *Empirical Software Engineering*, Springer, 3 (1), pp. 65-117, 1998.

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

[5] A. Chatzigeorgiou and A. Manakos, "Investigating the evolution of code smells in object-oriented systems", *Innovations in Systems and Software Engineering,* Springer, 10 (1), pp. 3-18, 2014.

[6] 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.

[7] S. Demeyer, S. Ducasse, and O. Nierstrasz, "Finding refactorings via change metrics", *15th Conference on Object-oriented Programming, Systems, Languages, and Applications* (OOPSLA'00). ACM, pp. 166-177, Minnesota, USA, 10 October 2000.

[8] L. Fernández and R. Peña, "A sensitive metric of class cohesion", *International Journal of Information Theories and Applications*, 13 (1), pp. 82-91, 2006.

[9] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, "Refactoring: Improving the Design of Existing Code", *Addison-Wesley*, 1999.

[10] B. Henderson-Sellers, "Object-Oriented Metrics Measures of Complexity", *Prentice-Hall*, 1996.

[11] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, "A quantitative evaluation of maintainability enhancement by refactoring", *18th International Conference on Software Maintenance* (ICSM'02), IEEE, Montreal, Canada, 3-6 October 2002.

[12] G. King and L. Zeng, "Logistic Regression in Rare Events Data Political Analysis", *Oxford Journal*, 9 (2), pp. 137-163, 2001.

[13] M. Kunz, R. R. Dumke, and A. Schmietendorf, "How to Measure Agile Software Development.", *Software Process and Product Measurement*, Lecture Notes in Computer Science, Springer, pp. 95-101, 2008.

[14] W. Li and S. M. Henry, "Maintenance metrics for the object oriented paradigm", *1st International Symposium on Software Metrics* (METRICS'93), IEEE, pp. 52-60, Baltimore, USA, 21-22 May 1993.

[15] M. Lippert and S. Roock, "Refactoring in Large Software Projects", *John Wiley & Sons*, 2006.

[16] H. Liu, Q. Liu, Z. Niu, and Y. Liu, "Dynamic and automatic feedbackbased threshold adaptation for code smell detection", *Transactions on Software Engineering*, IEEE, 42 (6), pp. 544–558, June 2011.

[17] T. De Marco, "Structured Analysis and System Specification", *Yourdon Press Computing Series*, 1979.

[18] L. Marg, L. C. Luri, E. O'Curran, and A. Mallett, "Rating Evaluation Methods through Correlation", *1st Workshop on Automatic and Manual Metrics for Operational Translation Evaluation* (MTE'14), Reykjavik, Iceland, 26 May 2014.

[19] R. Marinescu, "Detection strategies: metrics-based rules for detecting design flaws", *20th International Conference on Software Maintenance* (ICSM'04). IEEE, pp. 350-359, Chicago, USA, 11-14 September 2014.

[20] M. Ó Cinnéide, L. Tratt, M. Harman, S. Counsell, and I. H. Moghadam, "Experimental assessment of software metrics using automated refactoring", *6th International symposium on Empirical software engineering and measurement* (ESEM '12), ACM/IEEE, pp. 49-58, Lund, Sweden, 19-20 September 2012.

[21] F. Palomba, G. Bavota, M. D. Penta, F. Fasano, R. Oliveto, and A. De Lucia, "On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation" *Empirical Software Engineering*, 23 (3), pp. 1188–1221, 2017.

[22] F. Palomba, A. Panichella, A. Zaidman, R. Oliveto, and A. D. Lucia, "The scent of a smell: An extensive comparison between textual and structural smells," *Transactions on Software Engineering*, IEEE, 2017.

[23] F. Simon, F. Steinbruckner, and C. Lewerentz, "Metrics based refactoring", *5th European Conference on Software Maintenance and Reengineering* (CSMR'01), IEEE, Lisbon, Portugal, 14-16 March 2001.

[24] N. Tsantalis and A. Chatzigeorgiou, "Identification of extract method refactoring opportunities for the decomposition of methods", *Journal of Systems and Software*, Elsevier, 84 (10), pp. 1757-1782, 2011.

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

[26] N. Yoshida, K. Masataka, and I. Hajimu, "A cohesion metric approach to dividing source code into functional segments to improve maintainability", *16th European Conference on Software Maintenance and Reengineering* (CSMR'12), IEEE, pp. 365-375, Szeged, Hungary, 27-30 March 2012.