

Reducing Subjectivity in Code Smells Detection:

Experimenting with the Long Method

Sérgio Bryton
QUASAR/CITI/FCT
Universidade Nova de Lisboa
Caparica, Portugal
bryton@di.fct.unl.pt

Fernando Brito e Abreu
QUASAR/CITI/FCT
Universidade Nova de Lisboa
Caparica, Portugal
fba@di.fct.unl.pt

Miguel Monteiro
QUASAR/CITI/FCT
Universidade Nova de Lisboa
Caparica, Portugal
mmonteiro@di.fct.unl.pt

Abstract — Guidelines for refactoring, meant to improve software systems internal quality and widely acknowledged as one of software best practices, are still qualitative. As a result, the refactoring process is mostly subjective, and consequently non-automatable, prone to errors and unrepeatable. An example is the detection of the *Long Method* code smell. To address this problem, this paper proposes a technique to detect *Long Method* objectively and automatically, using a *Binary Logistic Regression* model calibrated by expert’s knowledge. The results of an experiment illustrating the use of this technique are then reported.

Keywords – Refactoring Process; Code Smells; Long Method; Binary Logistic Regression.

I. INTRODUCTION

Refactoring [1] is the process of modifying software implementation to improve its internal structure. Its objective is optimizing maintainability, reusability and/or efficiency [2], without altering software external behavior. Refactoring uses *code smells* (e.g. *Long Method*) as indicators of potential design flaws that harm maintainability and reusability [1, 3]. There is empirical evidence that some code smells are positively associated with class error probability in the evolution of object-oriented systems [2]. Several refactoring catalogues [1, 4-6] have been proposed which, among other things, characterize code smells and provide heuristics to detect them. Since the latter are commonsense rules most often described in natural language, the whole process of code smells detection is subjective, non-repeatable, error-prone and non-automatable [7]. This problem makes refactoring dependent on developer’s experience and is not cost-effective.

The idea of automating code smells detection by using metrics and tools is not new [8-11]. Such proposals usually require the setting-up of metrics’ thresholds, used by tools as decision criteria for the existence or absence of code smells. To the best of our knowledge, these thresholds are not consensual and have not been validated, thus leaving the detection process still in the realm of subjectivity. Consider, for instance, the proposal to detect the *God Class* smell, where the *AOFD*, *WMC* and *TCC* metrics are used [2]:

$$([\text{AOFD}] \text{ is in top } 20\%) \text{ AND } ([\text{AOFD}] > 4) \text{ AND } ([\text{WMC}] > 20) \text{ AND } ([\text{TCC}] < 33) \quad (1)$$

In section 4, we demonstrate that the detection of *Long Method* can be objective, deterministic and automatic. To achieve this, we define a mathematical model for this smell, by following a process inspired by the *MORE (Modularity-Oriented Refactoring)* method [12] and described in section 3. Thus, the expected contributions of this paper are:

- (1) The proposal of a mathematical model for detecting the *Long Method* code smell;
- (2) The feasibility demonstration of such a model.

II. SUBJECTIVITY IN DETECTING THE LONG METHOD

To fully appreciate how subjective the detection of this kind of code smell is, we quote some authors on the subject. Notice that not all authors dub this code smell the same way, but in essence they all refer to the same phenomenon.

On the *Long method*, by Fowler [1]:

- (1) Longer procedures are more difficult to understand;
- (2) Whenever we feel the need to comment something, we write a method instead;
- (3) The key is not method length but the semantic distance between what the method does and how it does it;
- (4) Look for comments to find the code to extract. This often signals this semantic distance;
- (5) Conditionals and loops also give signs for extractions.

On the *Bad routines*, by McConnell [3]:

- (1) Individual method or procedure is not invoked for a single purpose;
- (2) Routine has too many parameters. The upper limit for an understandable number of parameters is 7;
- (3) Routine reads and writes global variables.

On the *Routine size*, by McConnell [3]:

- (1) Accessor routines should be very short;
- (2) Issues such as depth of nesting, number of variables, and other complexity related considerations should dictate the length of the routine rather than imposing a length restriction per se;
- (3) A complex algorithm can grow up to 100 – 200 LOC;
- (4) Routines longer than 200 LOC decrease understandability.

The previous quotes are excerpts of two widely acknowledged books¹. Most probably they are used by many developers when they decide if a method is, or not, a *Long Method*. As we have seen, the decision can be arbitrary, since these criteria for detecting *Long Method* are not objective, therefore hard to automate. Nevertheless, we believe that they express useful knowledge and can be used as a starting point for more precise and objective criteria, thus assisting developers in their decision.

III. A MATHEMATICAL MODEL TO DETECT LONG METHOD

In this section we propose a mathematical model that can be used to increase the objectivity in detecting the *Long Method*. In concrete, we will evaluate if experts' judgment can be modeled by a *Binary Logistic Regression (BLR)* model. Such a model must be calibrated, based upon experts' classification on a training set. "Calibration" stands for determining the values of the model coefficients. After being calibrated, the inputs of such a model are a set of regressors (predictor variables), to be described in a following section.

A. Binary Logistic Regression

Introduction. Logistic regression (sometimes called the logistic model or logit model) is used for prediction of the probability of occurrence of an event by fitting data to a logistic curve [15]. The event here is for a method to be considered a surrogate of *Long Method*. *BLR* is a generalized linear model used for binomial regression. The logistic function is useful because it can take as an input any value from negative infinity to positive infinity, whereas the output is confined to values between 0 and 1.

Regressors. Like other forms of regression analysis, *BLR* makes use of several predictor variables (aka regressors) that may be either numerical or categorical. In our case the regressors are a set of methods' characteristics, expressed as code complexity metrics: *MLOC (Method Lines Of Code)* for length, *NBD (Nested Block Depth)* for nesting, *VG (cyclomatic complexity)* for control structure and *PAR (number of PARameters)* for interface [14]. Their choice was based on the fact that they are among those that can be found in most metric collection tools.

Model formulation. The dependent or outcome variable is named *IsLong* and represents the probability of a given method to be an instance of *Long Method*. In other words our *BLR*-based model expresses the probability (*IsLong*) of a method being a *Long Method*, based on its complexity characteristics. Our model is then the following:

$$IsLong = f(z) = \frac{1}{1 + e^{-z}} \quad (2)$$

where

$$z = \beta_0 + \beta_1 * VG + \beta_2 * MLOC + \beta_3 * NBD + \beta_4 * PAR \quad (3)$$

¹ - For instance, in Google Scholar [1] is cited in 3255 publications (on 10 April 2010).

The variable z represents the exposure to some set of risk factors, while $f(z)$ represents the probability of a particular outcome, given that set of risk factors (the method complexity metrics, in our case). The variable z is a measure of the total contribution of all the risk factors used in the model and is known as the *logit*.

Model parameters. The β_0 parameter is called the *intercept parameter* (value of z when all risk factors are 0) and β_1 through β_4 are the *regression coefficients* of the corresponding predictors. Each of these regression coefficients quantifies the contribution of the respective predictor. A positive regression coefficient means that the risk factor increases the probability of the outcome, while a negative regression coefficient means that the risk factor decreases the probability of that outcome. A large regression coefficient means that the risk factor strongly influences the probability of that outcome, while a near-zero regression coefficient means that that risk factor has little influence on the probability of that outcome.

The values of all β_i parameters are obtained by regression upon a training set of predictor and outcome values. While the former were obtained by running a metrics collection tool upon a set of classes, the latter (values of *IsLong* for each method in that set) were obtained by experts' opinion on the same methods.

B. Using the model – an example

Steps. Consider you have a calibrated model (more on that later in this paper). Then, to use this model, we must start by collecting the predictor values (regressors) on a given method, with a metrics collection tool. Finally, by instantiating the *BLR* model with the regressors' values, we obtain the probability of a method being an instance of *Long Method*, which is given by the value of *IsLong*.

The example. Consider, for instance, the *rtrim()* method taken from the *Apache Commons CLI* [13].

```
protected String rtrim(String s)
{
    // if the string is empty do nothing and return it
    if ((s == null) || (s.length() == 0))
    {
        return s;
    }
    // get the position of the last character in the
    string
    int pos = s.length();
    while ((pos > 0) &&
        Character.isWhitespace(s.charAt(pos - 1)))
    {
        --pos;
    }
    // remove everything after the last character
    return s.substring(0, pos)
}
```

TABLE I contains the corresponding values for the regressors.

TABLE I – PREDICTOR VALUES FOR THE RTRIM() METHOD

Method	MLOC	NBD	VG	PAR
rtrim	10	2	5	1

According to our calibrated model (see next section), the probability of the *rtrim()* method to be considered an instance of the *Long Method* code smell is considerably high (*IsLong* = 72,5%). Notice that this result is corroborated by the heuristics of Fowler [1], since even though the method is not very large, its author felt the need of including several comments for increased understandability.

IV. MODEL CALIBRATION AND VALIDATION

For calibration and validation purposes we selected the *Apache Commons CLI* open source project [13], that provides an API for processing command line interfaces. This case study was chosen because it is a widely used software and also because its size is not exaggerated for manual identification of *Long Method* instances. *Apache Commons CLI* was developed in Java and has 20 classes and 193 methods and constructors.

A. Identification of Long Method code smells by experts

In this study the three authors of this paper played the role of experts². To identify the *Long Method* we have followed Fowler’s heuristics reproduced in section II. Each method was independently inspected by each expert. We have only considered a method to be a *Long Method* in cases where we had a full match (all experts marked the method) or a partial match (2 out of 3 experts marked the method) followed by a consensus reaching discussion where the third expert also became “convinced”.

We found that 37 out of 193 methods (19%) should be considered as *Long Methods*. The latter were found in 13 out of 20 classes (65%).

B. Measuring the Apache Commons CLI

To collect the selected complexity metrics for each method, we have used an Eclipse plugin [16]. Some descriptive statistics of obtained results are presented in Table II. The MLOC minimum of zero may seem odd, but we have checked that it indeed occurs.

TABLE II - DESCRIPTIVE STATISTICS

	Min.	Max.	Mean	Std.Dev.	Skewness	Kurtosis
MLOC	0	69	6,50	11,095	3,243	12,565
NBD	0	15	1,50	1,331	6,165	55,544
VG	1	16	2,13	2,489	2,950	9,561
PAR	0	9	1,17	1,431	2,291	7,273

C. Validating the Regressors

Normality. When applying statistical tests, we prefer using parametric tests rather than their non-parametric counterparts, because the former are more powerful (come up with less false negatives). However, normality is a pre-condition for applying parametric tests. We have applied two normality tests (*Kolmogorov-Smirnov* and *Shapiro-Wilk*) to all metrics, as presented in Table III. None of them can be said to have a normal distribution.

TABLE III - NORMALITY TESTS FOR ALL METRICS

	<i>Kolmogorov-Smirnov</i> ^a			<i>Shapiro-Wilk</i>		
	Statistic	df	Sig.	Statistic	df	Sig.
MLOC	,297	193	,000	,564	193	,000
NBD	,388	193	,000	,421	193	,000
VG	,374	193	,000	,528	193	,000
PAR	,325	193	,000	,725	193	,000

a. Lilliefors Significance Correction

Correlation. We are interested in finding high correlations between each metric variable and *IsLong*, but we are also interested in finding low correlations among the metrics variables, to guarantee that they are independent from each other.

Since none of the variables has a normal distribution, we have used the non-parametric *Spearman’s rho* correlation coefficient as represented in Table IV. Notice that all correlations are significant, with a 99% confidence interval.

TABLE IV – SPEARMAN’S RHO CORRELATION COEFFICIENTS

		MLOC	NBD	VG	PAR	IsLong
MLOC	Correlation	,759	,750	,298	,659	
	Sig. (2-tailed)	,000	,000	,000	,000	
NBD	Correlation	,759	,913	,272	,846	
	Sig. (2-tailed)	,000	,000	,000	,000	
VG	Correlation	,750	,913	,303	,819	
	Sig. (2-tailed)	,000	,000	,000	,000	
PAR	Correlation	,298	,272	,303	,327	
	Sig. (2-tailed)	,000	,000	,000	,000	
IsLong	Correlation	,659	,846	,819	,327	
	Sig. (2-tailed)	,000	,000	,000	,000	

Although there is no widespread consensus among the statisticians regarding correlation strength, we will adopt the following ranges, proposed in [17].

TABLE V - CORRELATION STRENGTH CATEGORIZATION

Strength	Range
Very high	[90%, 100%]
High	[70%, 90%[
Moderate	[40%, 70%[
Low	[20%, 40%[
Negligible	[0%, 20%[

The results presented in Table IV show us that while all the other metrics have a moderate to high correlation with *IsLong*, *PAR* exhibit a low correlation, thus seeming to be the worst predictor among the chosen metric set.

Also notice the very high correlation between VG and NBD, which indicates a potential threat, if they present multicollinearity. The latter occurs when predictor variables are highly correlated among themselves.

Multicollinearity. The collinearity statistics in Table VI, allow us to consider all regressors as acceptable. In fact, the presence of multicollinearity is assumed for Tolerance values lower than 0,20 or VIF values higher than 5 [18].

² - We all have experience on teaching the topic of code smells

TABLE VI: MODEL COEFFICIENTS AND COLLINEARITY DIAGNOSIS

Model		Collinearity Statistics	
		Tolerance	VIF
1	(Constant)		
	MLOC	,244	4,102
	NBD	,369	2,708
	VG	,273	3,661
	PAR	,885	1,130

Model validation. According to Table VII, we can see that all regressors are significantly related to *IsLong*.

TABLE VII: VARIABLES NOT IN THE EQUATION FOR BLR

Step 0	Variables	Score	df	Sig.
	VG	103,607	1	,000
	MLOC	90,780	1	,000
	NBD	81,134	1	,000
	PAR	14,446	1	,000
	Overall Statistics	109,615	4	,000

According to Table VIII, we can see that the overall model is significant when all four independent variables are entered. The null hypothesis for the Omnibus test is that adding the predictors to the model has not significantly increased our ability to predict if the method is a *Long Method*.

TABLE VIII: OMNIBUS TESTS FOR BLR

Step 1	Step	Chi-square	df	Sig.
	Block	147,827	4	,000
	Model	147,827	4	,000

Goodness-of-fit analysis. According to Table IX, we can see that the R Squares presented in both tests give a rough estimate of the variance in *IsLong* that can be predicted from the combination of the four variables. The *Cox and Snell* test is usually an underestimate. The *-2 Log likelihood statistic* is quite small, meaning that the model predicts well the occurrence of *Long Methods*. The *Nagelkerke R Square* statistic corroborates this observation.

The *Hosmer-Lemeshow* goodness-of-fit test considers the null hypothesis that there is a linear relationship between the predictor variables and the log odds of the criterion variables. According to Table X we reject this hypothesis.

TABLE IX: MODEL SUMMARY FOR BLR

Step	-2 Log likelihood	Cox & Snell R Square	Nagelkerke R Square
1	40,809*	,535	,858

TABLE X: HOSMER AND LEMESHOW TEST FOR BLR

Step	Chi-square	df	Sig.
1	28,218	6	,000

According to Table XI, we can see that 98,7% of the methods which are not *Long Methods* are predicted correctly with this model, while 83,8% of the *Long Methods* were predicted correctly. In other words, we had 2 false positives and 6 false negatives. We have predicted 33 (31+2) *Long Methods*. We were wrong in just 2 cases, so our false positive rate is $2/33 = 6\%$. We have predicted that 160 (154+6) methods were not *Long Methods*. We were wrong in 6 cases. Thus, our false negatives rate is $6/160 = 4\%$.

TABLE XI: CLASSIFICATION TABLE FOR BLR

Observed	IsLong	Predicted		
		IsLong		Percentage Correct
		0	1	
Step 1	0	154	2	98,7
	1	6	31	83,8
	Overall Percentage			95,9

a. The cut value is ,500

Table XII: VARIABLES IN THE EQUATION FOR BLR

	B	S.E.	Wald	df	Sig.	Exp(B)	95.0% C.I. for Exp(B)		
							Lower	Upper	
Step 1	VG	,598	,327	3,353	1	,867	1,919	,999	3,451
	MLOC	-,057	,096	,362	1	,548	,944	,793	1,139
	NBD	4,701	1,247	14,305	1	,800	110,062	9,548	1268,650
	PAR	,486	,326	2,224	1	,395	1,626	,860	3,061
	Constant	-11,336	2,525	20,159	1	,001	,000		

According to Table XII, we can see that *NBD* and *VG* are significant predictors ($\alpha=0,10$) when all variables are considered together. Since *MLOC* and *PAR* were significant predictors when considered alone and are not significant when considered together, this suggests some correlation. The Wald statistic tests the unique contribution of each predictor in the context of the other predictors, from which we can see that *NBD* is the predictor which most contributes to our model.

Calibrated model. The *z* value required to calculate the probability (*IsLong*) of a method being a *Long Method* is then the one we can obtain by substituting the coefficients in equation (3) by the values in the B column of Table XII, as follows:

$$z = -11.336 + 0.598 * VG - 0.057 * MLOC + 4.701 * NBD + 0.486 * PAR \quad (4)$$

V. RELATED WORK

Li and Shatnawi [2] conducted an empirical study in which they have found that some code smells were positively associated with the class error probability in three error-severity levels. This finding supports the use of code smells as a systematic method to identify and refactor problematic classes in this specific context.

Several refactoring catalogues [1, 4-6] provide heuristics to detect code smells.

Mens and Tourwé [19], Simon et al.[8], Habra and Lopez [10] and Tahvildari et al. [9, 20, 21] corroborate the opinion that metrics can identify potential refactorings and estimate the refactoring effect.

Simon et. al. [8] agree with the difficulty of identifying where to apply each refactoring, and propose object-oriented cohesion-based metrics to mitigate this problem.

Gronback [11] demonstrates how metrics can be used to identify some code smells by using thresholds.

Our approach eliminates the subjectivity of the detection of code smells, either based on heuristics or based on thresholds that are not consensual nor have been validated. Furthermore, our approach allows the ranking of code smells, based upon the value of the probability *IsLong*, which can be used, for instance, in the prioritization of refactoring actions. We have also shown that it is possible to assess the power of the prediction being used, in terms of the expected false positive/negative rates.

VI. CONCLUSIONS AND FUTURE WORK

A. Conclusions

We have demonstrated through a case study that the *BLR* model can be used to detect instances of *Long Method* in a more objective way, thus opening the way for its automation. In a nutshell, the *BLR* model requires calibration before being used. For achieving that, a set of experts should reach a consensus on which are the *Long Methods* in a training set of methods. Using a metrics collection tool, the regressors (code complexity metrics) are calculated and finally a statistics tool (e.g. SPSS, SAS or R) should be used to calculate the *BLR* coefficients. The *BLR* model is then ready for calculating the probability of a given method being an instance of *Long Method*, using as input the code complexity metrics, which are easily collectable.

Obviously, the larger the training set and the number of experts validating the preliminary classification, the better is expected to be the power of prediction. We have shown that using a small set of widely used complexity metrics, with a moderate sized training set and a small number of experts, we were able to build a model that predicted correctly almost 99% of the methods which are *Long Methods* and 84% of the methods which are not, with a false positive rate of 6% and a false negative rate of 4%. The most significant predictor in our model was *NBD*, while *MLOC* and *PAR* are not significant when used with the remainder variables. The effect of *VG*, although significant, is much smaller than that of *NBD*. The *PAR* is the least important predictor of *IsLong*, either alone or within the remainder variables in the model.

We now sum up the strengths and weaknesses of our process to detect this code smell.

Strengths. The process strengths are the following:

- (1) Benefits from the expert's knowledge;
- (2) Benefits from the power of statistical techniques;
- (3) Process is objective and deterministic;
- (4) Enables the automation of the process for detecting *Long Method*;

- (5) Allows the ranking of instances of *Long Method* by probability;
- (6) Enables the quantitative evaluation of the benefits of refactoring;
- (7) We can predict how many *Long Methods* can be detected;
- (8) We can predict how many false positives/negatives can be detected.

Weaknesses. The process weaknesses are as follows:

- (1) Model calibration requires the availability of code smells experts;
- (2) The size and type of the training set may influence the results;
- (3) The model may still lead to false positives or false negatives.

Our model cannot be generalized, since its calibration was performed upon a single project and its detection ability was only assessed upon the same project. However, the goal of demonstrating that the *Long Method* code smell can be detected automatically and objectively, grounded on expert's knowledge and statistical analysis, has been fully achieved.

B. Future work

We intend to conduct experiments and propose a general model for the detection of the *Long Method* code smell, by experimenting with other code complexity metrics and larger training sets, as well as performing the detection outside the training set, using a Jack-Knifing approach.

We are also researching if the approach we used to detect the *Long Method* code smell can be applied to detect other code smells. We expect to present the results of this effort shortly.

We believe that in the future IDEs will embody code smells detection models such as the one presented in this paper.

ACKNOWLEDGMENT

The work presented herein was partly supported by the *VALSE project* of the CITI research center within the Department of Informatics at FCT/UNL in Portugal.

REFERENCES

- [1] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 2000.
- [2] W. Li, and R. Shatnawi, "An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution", Journal of Systems and Software, vol. 80, num. 7, 2007, pp. 1120-1128, doi: 10.1016/j.jss.2006.10.018.
- [3] S. McConnell, Code Complete - A Practical Handbook of Software Construction. 2nd edition, Microsoft Press, 2004.

- [4] M. P. Monteiro, and J. M. Fernandes, "Towards a Catalogue of Refactorings and Code Smells for AspectJ", LNCS - Transactions on Aspect-Oriented Software Development I, vol. 3880, 2006, pp. 214-258.
- [5] J. Kerievsky, Refactoring to Patterns, Addison-Wesley, 2004.
- [6] R. Laddad, Aspect-Oriented Refactoring, Addison-Wesley, 2006.
- [7] S. Bryton: "Strengthening Refactoring: Towards Software Evolution with Quantitative and Experimental Grounds". Proceedings of the 4th International Conference in Software Engineering Advances (ICSEA'2009), Porto, Portugal, September 20-25, 2009.
- [8] F. Simon, F. Steinbrückner, and C. Lewerentz: "Metrics Based Refactoring". Proceedings of the CSMR'2001, Lisbon, Portugal, March, 2001, pp. 30-38.
- [9] M. Salehie, S. Li, and L. Tahvildari: "A Metric-Based Heuristic Framework to Detect Object-Oriented Design Flaws". Proceedings of the 14th International Conference on Program Comprehension (ICPC'06), Athens, Greece, 2006, pp. 159-168, doi: 10.1109/ICPC.2006.6.
- [10] N. Habra, and M. Lopez: "On the Use of Measurement on Software Restructuring". Proceedings of the International ERCIM Workshop on Software Evolution, Lille, France, 2006, pp. 81-88.
- [11] R. C. Gronback. *Software Remodeling: Improving Design and Implementation Quality, Using Audits, Metrics and Refactoring in Borland Together ControlCenter*. White Paper, Borland. Online at http://www.borland.com/resources/en/pdf/white_papers/, 2003.
- [12] S. Bryton, and F. B. Abreu: "Modularity-Oriented Refactoring". Proceedings of the 12th European Conference on Software Maintenance and Reengineering (CSMR'2008), Athens, Greece, Jan, 2008, pp. 294-297, doi: 10.1109/CSMR.2008.4493330.
- [13] A. S. Foundation. *Apache Commons CLI*. <http://commons.apache.org/cli/>, accessed on August 2009.
- [14] B. Henderson-Sellers, Object-Oriented Metrics - Measures of Complexity, Prentice Hall PTR, Upper Saddle River, NJ, EUA, ISBN: 0-13-239872, 1996.
- [15] N. Leech, K. Barrett, and G. A. Morgan, SPSS for Intermediate Statistics: Use and Interpretation. 3rd edition, Psychology Press ISBN: 978-0-8058-6267-6, 2007.
- [16] F. Sauer. *Metrics plugin for Eclipse*, version 1.3.6, available on <http://metrics.sourceforge.net>, accessed on August 2009.
- [17] M. H. Pestana, and J. N. Gageiro, *Análise de Dados para Ciências Sociais: A Complementaridade do SPSS*. 5th edition, Edições Sílabo, Lisboa, ISBN: 978-972-618-498-0, 2008.
- [18] A. Field, *Discovering Statistics Using SPSS*. 3rd edition, SAGE Publications, London, ISBN: 978-1-84787-906-6, 2009.
- [19] T. Mens, and T. Tourwé, "A survey of software refactoring", IEEE Transactions on Software Engineering, vol. 30, num. 2, 2004 pp. 126 - 139, doi: 10.1109/TSE.2004.1265817.
- [20] L. Tahvildari, and K. Kontogiannis, "A Metric-Based Approach to Enhance Design Quality Through Meta-Pattern Transformations", 7th European Conference on Software Maintenance and Reengineering (CSMR'03), 2003, pp. 183-192, doi: 10.1109/CSMR.2003.1192426.
- [21] L. Tahvildari, "Quality-Driven Object-Oriented Re-engineering Framework", 20th International Conference on Software Maintenance (ICSM'04), 2004, pp. 479-483, doi: 10.1109/ICSM.2004.1357839.