2010 Seventh International Conference on the Quality of Information and Communications Technology

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

Abstract — **Guidelines for refactoring are meant to improve software systems internal quality and are widely acknowledged as among software's best practices. However, such guidelines remain mostly qualitative in nature. As a result, judgments on how to conduct refactoring processes remain mostly subjective and therefore non-automatable, prone to errors and unrepeatable. The detection of the *Long Method* code smell is an example. 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 reported.**

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

I. INTRODUCTION

Refactoring [1] is the process of modifying the structure and programming style of a software system in view of improving maintainability, reusability and/or efficiency and without affecting its external behavior [2]. Decisions for when to resort to refactoring are often based on *code smells* (e.g. *Long Method*), i.e., 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, 5] 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 [6]. 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 [7-10]. 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 *God Class*, i.e., a large non-cohesive class that has far too much responsibility [11], where the *Weighted Methods per Class* (WMC), *Access of Foreign Data* (AOFD), and *Tight Class Cohesion* (TCC) metrics are used, with the following threshold values [2]:

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

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

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

This paper is organized as follows: we overview the subjectivity problem in detecting the *Long Method* in the next section; we present our contributions in sections III and IV; we then survey the related work in section V and finally draw our conclusions and perspectives for future work in the last section.

II. SUBJECTIVITY IN DETECTING THE LONG METHOD

To fully appreciate how subjective the detection of this kind of code smell is, we quote some widely cited authors on the subject. Note than not all authors name 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]:

978-0-7695-4241-6/10 $26.00 © 2010 IEEE
DOI 10.1109/QUATIC.2010.60

IEEE computer society

(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 McConnel [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 books[1]. Most probably they are used by many developers when they decide if a method is, or not, a *Long Method*. On the basis of the above cited guidelines, we believe decisions can be arbitrary, since criteria for detecting *Long Method* are not objective and therefore hard to automate. Nevertheless, we also believe they express useful knowledge and can be used as a starting point for more precise and objective criteria, thus assisting developers in their decisions.

III. A MATHEMATICAL MODEL TO DETECT LONG METHOD

In this section, we propose and describe a mathematical model, based upon *Binary Logistic Regression* (*BLR*), to support the automated detection of *Long Method* instances in a given source code component.

BLR requires an initial *calibration* phase, i.e., the determining of the values of the model coefficients on the basis of a group of experts' classification on a training set. In this context, *experts* are people with significant hands-on experience in detecting, classifying and removing code smells in software systems. In the study, all three authors separately played the role of expert[2]. An assessment of the extent to which the experts' judgment can be modeled by the *BLR* model is presented.

After being calibrated, the *BLR* model is fed with a set of regressors (predictor variables), as described in the next section.

A. Binary Logistic Regression

Introduction. Logistic regression (aka logistic model or logit model) is used for prediction of the probability of occurrence of an event by fitting data to a logistic curve [13]. Here, the event is deeming a method 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 regressors (aka predictor variables) that may be either numerical or categorical. In this 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 their general availability in metric collection tools.

Model formulation. The dependent or outcome variable *IsLong* represents the probability of a given method to be an instance of *Long Method*, i.e., the *BLR*-based model expresses the probability (*IsLong*) of a method being a *Long Method* on the basis of its complexity characteristics. The model is thus the following:

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

where

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

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 (method complexity metrics in this case). 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. 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 [15] 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. Assume you have a calibrated model (section IV). Then, to use this model, we start by collecting the regressors (predictor values) 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* [16].

[1] - For instance, in Google Scholar Fowler [1] is cited in 3420 publications (on 28 June 2010).
[2] - All authors have experience on teaching the code smells topic.

```
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 the calibrated model (section IV), the probability of the *rtrim*() method to be considered an instance of the *Long Method* code smell is considerably high (*IsLong* = 72,5%). Note 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 [16], 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

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 *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 [15]. 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

	Kolmogorov-Smirnov[a]			*Shapiro-Wilk*		
	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 of each other.

Since none of the variables has a normal distribution, we use the non-parametric *Spearman's rho* correlation coefficient as represented in Table IV. Note 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 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* exhibits low correlation and therefore seems to be the worst predictor among the chosen metric set.

Also note 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].

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*

			Score	df	Sig.
Step 0	Variables	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*

		Chi-square	df	Sig.
Step 1	Step	147,827	4	,000
	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[a]	,535	,858

TABLE X: HOSMER AND LEMESHOW TEST FOR BLR

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

According to Table XITable , 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 the 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, the false negatives rate is 6/160 = 4%.

TABLE XI: CLASSIFICATION TABLE FOR BLR

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

a. The cut value is ,500

According to Table XII, we can see that *NBD* and *VG* are significant predictors (α=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 that most contributes to the *BLR* model.

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	,067	1,819	,959	3,451
	MLOC	-,057	,096	,362	1	,548	,944	,783	1,139
	NBD	4,701	1,247	14,205	1	,000	110,062	9,548	1268,650
	PAR	,486	,326	2,224	1	,136	1,626	,858	3,081
	Constant	-11,336	2,525	20,150	1	,000	,000		

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, 5, 19] provide heuristics to detect code smells.

Mens and Tourwé [20], Simon et al.[7], Habra and Lopez [9] and Tahvildari et al. [8, 21, 22] corroborate the opinion that metrics can identify potential refactorings and estimate the refactoring effect.

Simon et. al. [7] agree with the difficulty of identifying where to apply each refactoring, and propose object-oriented cohesion-based metrics to mitigate this problem.

Gronback [10] demonstrates how metrics can be used to identify some code smells by using thresholds.

Our approach significantly reduces subjectivity in 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 show it is possible to assess the power of the prediction used, in terms of expected false positive/negative rate.

VI. CONCLUSIONS AND FUTURE WORK

A. Conclusions

We demonstrate 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 computed and finally a statistics tool (e.g. SPSS, SAS or R) is used to calculate the *BLR* coefficients. The *BLR* model is then ready to estimate the probability of a given method being an instance of *Long Method,* using as input the code complexity metrics, which are easily collectable.

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 around 84% of the methods which are *Long Methods* and 99% 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.

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 (to improve prediction power), as well as performing the detection outside the training set, using a Jack-Knifing approach. Among other things, and for the sake of external validity, we want to assess if our prediction model generalizes appropriately across different types of application, or does it need to be recalibrated.

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.

Logistic regression is a mathematically sound approach, but if a limited number of experts participate, subjectivity is not removed but rather shifted to the experts' opinion, because the constructed regression model will somehow reflect their specific judgment, severely limiting the ability to generalize the results. Increasing the number of experts will cancel that individual bias introduced by experts, but will surely introduce classification conflicts. In our experiment, consensus among experts was relatively easy in most situations. However, for this approach to be more systematic, we plan to adopt a formal consensus technique, to avoid domination effects, majority vote solution (if the number of experts is odd) or ties (if it is even). Formal consensus is the least violent decision-making process [23].

We plan to build an Eclipse plugin embodying code smells detection models, including the one presented in this paper. Such a plugin is planned to support a distributed formal consensus technique. The latter would be somehow similar to the one available to program committee members in the final review phase, to reach an agreement on paper scores, as provided by web-based conference supporting tools, such as *EasyChair* [24]. With such a facility, and a periodic recalibration of the logistic regression models (one per each code smell), all power users will contribute to their tuning.

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, n. 7, pp. 1120-1128, Elsevier Science, ISSN:0164-1212, 2007.

[3] S. McConnell, *Code Complete - A Practical Handbook of Software Construction*, 2nd ed., 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, pp. 214-258, Springer, 2006.

[5] J. Kerievsky, *Refactoring to Patterns*, Addison-Wesley, 2004.

[6] S. Bryton and F. Brito e Abreu, "Strengthening Refactoring: Towards Software Evolution with Quantitative and Experimental Grounds", in proceedings of the *Fourth International Conference on Software Engineering Advances (ICSEA 2009)*, pp. 570-575, Porto, Portugal, 2009.

[7] F. Simon, F. Steinbrückner, and C. Lewerentz, "Metrics Based Refactoring", in proceedings of the *CSMR'2001*, pp. 30-38, P. Sousa and J. Ebert (Eds.), Lisbon, Portugal, 2001.

[8] M. Salehie, S. Li, and L. Tahvildari, "A Metric-Based Heuristic Framework to Detect Object-Oriented Design Flaws", in proceedings of the *14th International Conference on Program Comprehension (ICPC'06)*, pp. 159-168, Athens, Greece, 2006.

[9] N. Habra and M. Lopez, "On the Use of Measurement on Software Restructuring", in proceedings of the *International ERCIM Workshop on Software Evolution* (co-located with / organized by ERCIM Consortium), pp. 81-88, L. Duchien, M. D'Hondt, and T. Mens (Eds.), Lille, France, 2006.

[10] R. C. Gronback, "Software Remodeling: Improving Design and Implementation Quality, Using Audits, Metrics and Refactoring in Borland Together ControlCenter", White Paper, Borland, 2003.

[11] A. J. Riel, *Object-Oriented Design Heuristics*, Reading, MA, USA, Addison-Wesley Publishing Company, Reading, MA, USA, ISBN:0-201-63385-X, 1996.

[12] S. Bryton and F. Brito e Abreu, "Modularity-Oriented Refactoring", in proceedings of the *12th European Conference on Software Maintenance and Reengineering (CSMR'2008)*, pp. 294-297, Athens, Greece, 2008.

[13] N. Leech, K. Barrett, and G. A. Morgan, *SPSS for Intermediate Statistics: Use and Interpretation*, 3rd ed., Psychology Press ISBN:978-0-8058-6267-6, 2007.

[14] B. Henderson-Sellers, *Object-Oriented Metrics - Measures of Complexity*, Upper Saddle River, NJ, USA, Prentice Hall PTR, Upper Saddle River, NJ, USA, ISBN:0-13-239872, 1996.

[15] *Metrics Plugin for Eclipse*, version 1.3.6, 2009. Available: http://metrics.sourceforge.net. [Accessed: May 2010].

[16] A. S. Foundation, "Apache Commons CLI". Available: http://commons.apache.org/cli/. [Accessed: May 2010].

[17] M. H. Pestana and J. N. Gageiro, *Análise de Dados para Ciências Sociais: A Complementaridade do SPSS*, 5th ed., Lisboa, Edições Sílabo, Lisboa, ISBN:978-972-618-498-0, 2008.

[18] A. Field, *Discovering Statistics Using SPSS*, 3rd ed., London, SAGE Publications, London, ISBN:978-1-84787-906-6, 2009.

[19] R. Laddad, *Aspect-Oriented Refactoring*, Addison-Wesley, 2006.

[20] T. Mens and T. Tourwé, "A Survey of Software Refactoring", *IEEE Transactions on Software Engineering*, vol. 30, n. 2, pp. 126-139, February 2004, 2004

[21] L. Tahvildari and K. Kontogiannis, "A Metric-Based Approach to Enhance Design Quality Through Meta-Pattern Transformations", in proceedings of the *7th European Conference on Software Maintenance and Reengineering (CSMR'03)*, pp. 183-192, Benevento, Italy, 2003.

[22] L. Tahvildari, "Quality-Driven Object-Oriented Re-engineering Framework", in proceedings of the *20th International Conference on Software Maintenance (ICSM'04)*, pp. 479-483, Chicago, USA, 2004.

[23] C. T. L. Butler and A. Rothstein, *On Conflict and Consensus: A Handbook on Formal Consensus Decisionmaking*, 3rd ed., Creative Commons, 2007.

[24] *EasyChair Conference System*, University of Manchester, Manchester, UK, 2010. Available: http://www.easychair.org. [Accessed: May 2010].