Detecting God Methods with Concern Metrics

An Exploratory Study

Juliana Padilha[1], Eduardo Figueiredo[1], Cláudio Sant'Anna[2], Alessandro Garcia[3]

[1] Computer Science Department, Federal University of Minas Gerais (UFMG), Brazil
[2] Computer Science Department, Federal University of Bahia (UFBA), Brazil
[3] Informatics Department, Pontifical Catholic University of Rio de Janeiro (PUC-RIO), Brazil

{juliana.padilha, figueiredo}@dcc.ufmg.br, santanna@dcc.ufba.br, afgarcia@inf.puc-rio.br

Abstract— **Software metrics have been traditionally used to evaluate the modularity of software systems and to detect code smells, such as God Method. Code smells are symptoms that may indicate something wrong in the system code. God Method represents a method that has grown too much. It tends to centralize the functionality of a class. Recently, concern metrics have also been proposed to evaluate software maintainability. While traditional metrics quantify properties of software modules, concern metrics quantify properties of concerns, such as scattering and tangling. Despite being increasingly used in empirical studies, there is a lack of empirical knowledge about the usefulness of concern metrics to detect code smells. This paper goal is to report the results of an exploratory study which investigates whether concern metrics provide useful indicators to detect God Method. In this study, a set of 47 subjects from two institutions have analyzed traditional and concern metrics aiming to detect instances of this code smell in a system. The study results indicate that elaborated joint analysis of both traditional and concern metrics is often required to detect God Method. We conclude that new focused metrics may be required to support detection of smelly methods.**

Keywords— Metrics; code smells; separation of concerns.

I. Introduction

The modularization of the driving design concerns is a key factor to achieve maintainable software systems [15, 19]. A concern is any important property or area of interest of a system that we want to treat in a modular way [21]. However, certain concerns, called crosscutting concerns [15], might end up being scattered and tangled with each other. Security, persistence, and exception handling are examples of typical crosscutting concerns found in many software systems. The inadequate separation of concerns degrades design modularity and may lead to design flaws, such as code smells [6, 11, 12]. Detection of these code smells by programmers is far from trivial and usually requires quantitative support [16, 17].

Software metrics are key means for assessing software modularity and detecting design flaws [3, 7]. The community of software metrics has traditionally explored quantifiable module properties, such as class coupling, cohesion, and interface size, in order to identify code smells in software systems [3, 8, 16, 17]. For instance, Marinescu [17] relies on traditional metrics to detect code smells. However, some code smells are often a direct result of poor separation of concerns and traditional module-driven metrics cannot detect them.

On the other side, a growing number of concern metrics have been proposed [5, 6, 23] aiming to quantify the key concern properties, such as scattering and tangling. Differently from traditional metrics, concern metrics quantify properties of concerns realized by several modules in the code [10]. Concern metrics have been applied with different purposes and used in several empirical studies [4, 11, 13, 14]. They are used, for instance, to compare aspect-oriented and object-oriented programming techniques [4, 11, 13] and to identify crosscutting concerns that should be refactored [6]. However, we still lack empirical knowledge on the effectiveness of concern metrics to support code smell detection.

In this context, this paper presents an exploratory study that investigates the efficacy of concern metrics on the identification of one code smell [12], named God Method, in a software system. God Method represents a method that has grown too much and tends to centralize the functionality of a class [12]. This exploratory study involved 47 subjects, which were divided in three groups. Subjects of each group relied on the analysis of one of three different sets of metrics (Section II): (i) only *traditional metrics*, (ii) only *concern metrics*, and (iii) both traditional and concern metrics, called *hybrid metrics* from now on. This study focuses on a two-dimension analysis comparing the trade-offs on the accuracy and time efficiency of code smell detection (Section III). To analyze the accuracy, we compare methods identified as suspects of exhibiting the code smell with the code smell reference list provided by the actual developers of the system. We also assess time efficiency based on the recorded time spent by each subject in the experimental tasks.

Our overall results (Section IV) suggest that current concern metrics does not always contribute to the detection of God Method. In fact, we observed that only four concern metrics can offer useful indicators to detect this code smell. Other lessons learned are: (i) detection of God Method requires joint analysis of several traditional and concern metrics, and (ii) the use of more metrics not necessarily leads to longer analysis. Section V discusses the study constraints. Section VI concludes this paper and points out directions for future work.

II. BACKGROUND

This section introduces relevant background topics for this paper. Traditional and concern metrics are presented in Section II.A. Section II.B describes the code smell we investigate in this study. Section II.C discusses previous work that relies on metrics to detect code smells.

A. Software Metrics

Software metrics have played an important role in understanding and analyzing modularity of software systems [3, 7, 16]. For the purpose of this study, software metrics can be divided in three sets: (a) traditional metrics, (b) concern metrics, and (c) hybrid metrics, which includes both traditional and concern metrics. We selected some of the most widely used metrics to analyze in this study. The selected traditional metrics includes object-oriented (OO) metrics proposed by Chidamber and Kemerer [3] and well-documented metrics in the software engineering literature [7]. Table I summarizes the traditional metrics we used in this study. Their detailed definitions can be found elsewhere [3, 7].

TABLE I. DEFINITIONS OF TRADICITIONAIS METRICS

Metric	Definition
Coupling between Objects (CBO)	It counts the number of classes which a class calls methods or an access attributes.
Lack of Cohesion in Methods (LCOM)	It counts pairs of methods that do not access common attributes by pairs that do access.
Lines of Code (LOC)	It counts the total number of lines of code.
Number of Attributes (NOA)	It counts the number of attributes defined in a class.
Number of Methods (NOM)	It is the number of methods defined in a class.
Number of Parameters in Methods (PAR)	It is the number of methods defined by each method in a class.
Weighted Methods per Class (WMC)	It counts the number of methods and their parameters in a class
Cyclomatic Complexity (CYCLO)	It counts the number of flows in a method. It is incremented each time a branch occurs.

This paper focuses on the evaluation of concern metrics since traditional metrics have already been studied by previous work aiming to detect code smells [16, 17]. Concern metrics are defined to capture modularity properties associated with the realization of concerns in software artifacts [6, 23]. Concern metrics rely on a mapping between concerns and code elements [9, 11]. This mapping consists of assigning a concern to the corresponding code elements that realize it. This study relies on concern mappings performed by developers or domain experts on the target system.

Table II presents a brief definition of the concern metrics evaluated in this paper. These concern metrics focus on quantifying the degree of concern scattering and tangling. Concern scattering is defined as the degree to which a concern is spread over the code elements, while concern tangling represents the degree to which a concern is mixed up with other concerns in a module implementation [15, 21]. A more detailed description and discussion of these metrics can be found elsewhere [6, 9, 13, 23]. These concern metrics were selected for evaluation in this paper because they have successfully been used in a number of studies related to software modularity [6, 9, 11, 13]. However, no systematic study has been performed to evaluate whether these concern metrics support the detection of the God Method code smell.

TABLE II. DEFINITIONS OF CONCERN METRICS

Metric	Definition
Concern Diffusion over Operations (CDO)	It counts the number of methods whose main goal is to implement a concern.
Concern Diffusion over Attributes (CDA)	It counts the number of attributes whose main goal is to implement a concern.
Number of Concerns Lines of Code (LOCC)	It counts the number of lines of code whose main goal is to implement a concern.
Concern Diffusions over LOC (CDLOC)	It counts the number of transition points for each concern through the lines of code. Transition points are points in the code where there is a “concern switch”.
Number Concerns per Component (NCC)	It counts the number of concern implemented by each class.
Number Concerns per Method (NCO)	It counts the number of concern implemented by each method in a class.

B. God Method Code Smell

Code smells are a mean to diagnose symptoms that may be indicative of something wrong in the system code [12]. Previous work has shown that code smells might be found by means of traditional metrics [16, 17]. This paper investigates the use of concern metrics to detect one code smell, named God Method [12]. This code smell, described below, was chosen because previous studies have related it with poor modularization of concerns [17].

The God Method code smell represents a method that has grown too much [12]. In general, the longer a method is, the more difficult it is to understand. This code smell tends to centralize the functionality of a class [12]. In a different perspective, we can say that God Method is a method that implements many concerns and, so, it has many responsibilities.

C. Metrics-based Detection of Code Smells

Metrics has been historically used to detect code smells [16, 17]. Marinescu [17] proposed the use of strategies composed of traditional metrics for detecting code smells. His work observed that multiple metrics are required to capture all factors in the code smell definition. Its evaluation of code smell detection indicates an accuracy of about 60% for most code smells. His study relied on several traditional metrics also used in our study, but it has not used concern metrics.

Unlike Marinescu's work, several studies have used concern metrics to assess diverse attributes of software systems, such as modularity [23], instability [11, 14] and error-proneness [6, 8]. For instance, Eaddy and his colleagues [6] have carried out three experiments to evaluate the usefulness of concern metrics to identify error-prone modules. Other studies [11, 14] related on concern metrics to support the comparison of aspect-oriented and object-oriented decompositions.

Different from our work, however, these studies implicitly assume that concern metrics are reliable indicators for their respective aims.

III. Study Setting

This study aims at evaluating the accuracy of concern metrics to detect God Method. Our study relies on traditional metrics as baseline. Section III.A discusses the research questions of this study. Section III.B introduces the software system we used, named MobileMedia [11]. Sections III.C and III.D present the reference list of code smell and background information for the subjects that took part in this study, respectively. Finally, Section III.E explains the tasks assigned to each subject.

A. Research Questions

The goal of this study is to find out whether concern metrics are appropriate means to detect God Method. Therefore, the general research question we aim to answer can be formulated as follows.

Do concern metrics support God Method detection?

In order to answer this research question, we conducted an investigation with 47 subjects. Specific questions that can be derived from the general question are: (i) How accurate do concern metrics perform in comparison with traditional ones to detect God Method? (ii) Is there a specific metric that accurately detect God Method? (iii) Is there a combination of metrics that increase the accuracy of identifying God Method? (iv) Can a larger set of metrics make the task of identifying God Method more time consuming?

B. The MobileMedia System

Our study involved the last version of the MobileMedia system [11]. This system is a software product line (SPL) for applications that manipulate photo, music and video on mobile devices, such as mobile phones. It is an open source project with about 3.5 KLOC. The concerns we considered in MobileMedia to apply concern metrics are: (a) *Sorting* allows counting the number of times a particular media was viewed by the user and sorting media by the number of views; (b) *Favorites* allows the user to define a particular media as favorite and to visualize favorite media; (c) *Exception Handling* implements a mechanism to deal with events that change the normal flow of execution; (d) *Security* allows passwords to be associated with media albums; and (e) *Persistence* refers to the ability of the application to retain data between executions. This system was selected because it has been previously used in other modularity-related studies [4, 8, 11]. In addition, we have access to its developers and otherwise experts and, so, we were able to recover a reference list of its actual code smells.

C. God Method Reference List

Before conducting the study, we performed a systematic code analysis of MobileMedia aiming to determine which methods are God Methods (Section II.B). In addition to ourselves, we also relied on two experts in this system to help us building a reference list for each analyzed code smell. These experts participated on the development, maintenance, or assessment of the system. Our goal was to detect actual instances of God Method in MobileMedia. Table III presents the seven methods in the reference list for this code smell.

TABLE III. God Method Reference List

Methods smelling God Method	
MainUIMidlet.startApp	MediaListController.showMediaList
AlbumController.handleCommand	MusicPlayController.handleCommand
MediaController.handleCommand	PhotoViewController.handleCommand
MediaController.showImage	

D. Background of Subjects

This study involved 47 subjects from two different institutions. Subjects from the first institution were 22 undergraduate and 15 post-graduated (Msc and PhD) students. Subjects from the second institution were all 10 post-graduated (Msc and PhD) students. The study was performed using the OO design of MobileMedia. We organized subjects in such a way that each group worked only with one set of metrics: traditional metrics, concern metrics, or hybrid metrics. Subjects were grouped trying to balance their background knowledge. As a result, all 47 subjects detected God Method, but using different sets of metrics. Further details about the distribution of subjects are available at the project website [1].

Before start running the experiment, we used a background questionnaire to acquire previous knowledge of each subject. Table IV summarizes knowledge that subjects claimed to have in the background questionnaire. They answered questions about their previous experience with respect to Class Diagram, Java Programming, Software Metrics, and Work Experience. Subjects are named S1 to S47. The last three columns in Table IV show the subjects who claimed to have knowledge in a particular skill. There are subjects that do not appear in a row because they do not have experience in that particular topic. For instance, the following subjects do not have work experience: S1, S21, S23, S32, S36, S43, S44, and S45.

TABLE IV. Background Data Subjects

		Traditional	**Concern**	**Hybrid**
Knowledge	Class Diagram	S1-S16	S17-S32	S33-S47
	Java Program.	S1-S16	S17-S32	S33-S47
	Work Experience	S2-S16	S17-S20, S22, S24-S31	S33-S35, S37-S42, S46, S47
	Measurement	S1-S16	S17-S32	S33-S47

We can observe in Table IV that all subjects have at least basic knowledge in Class Diagram, Java programming, and measurement. In fact, we asked subject to indicate their level of knowledge by choosing one of the following options: none, few, moderate, and high experience (details at [1]). However, after analyzing the results, we observe that the level of knowledge does not have a high impact on the main conclusions. Therefore, we do not control this variable in this study. We only use this data to make a fair distribution of subjects among the groups of metrics.

E. Experimental Tasks

The study was preceded by a 30-minute training session to allow subjects to familiarize themselves with the evaluated metrics and the target code smell. Each subject had to detect God Method instances. After the training session, each subject received a document containing: (i) a brief explanation and a partial view of the system design as a class diagram and (ii) a description of the concerns involved in the respective analyzed system. The document also described steps and guidelines that subjects should follow, the questions they should answer, and information they should register.

In addition, we provided subjects with the results of the metrics in the respective system under analysis. In order to identify God Methods, each group of subjects (traditional group, concern group or hybrid group, for short) only had access to the results of metrics they were assigned to. Subjects have no access to the system source code. We also asked each subject to explain which metrics were useful for detecting the code smell and which ones were not useful.

IV. Results and Analysis

This section presents the results of our experiments. Section IV.A introduces two metrics, recall and precision, we used in the analysis of the results. Section IV.B discusses joint analysis of traditional and concern metrics performed by subjects. Section IV.C analyzes the accuracy of concern metrics compared to traditional metrics. Section IV.D tries to uncover a specific metric that seems most appropriate to detect this code smell. Section IV.E focuses on combinations of metrics. Finally, Section IV.F draws an analysis of time per set of metrics.

A. Recall and Precision

We rely on three metrics, namely True Positives (TP), False Positives (FP), and False Negatives (FN), collected from data that subjects provided us. True Positives and False Positives quantify the number of correctly and wrongly identified code smells by a subject. False Negatives, however, quantify the number of code smells a subject missed out. Based on these metrics, we quantify recall and precision, presented below, to support our analysis. Recall (R) measures the fraction of relevant methods listed by a subject. Relevant methods are methods that appear in the reference list. Precision (P) measures the ratio of correctly detected code smells by the total methods a subject listed.

$$\text{Recall (R)} = \frac{TP}{TP + FN} \qquad \text{Precision (P)} = \frac{TP}{TP + FP}$$

We focus our discussion mainly on recall because it is a measure of completeness. That is, high recall means that the subject was able to identify most code smells in the system. High precision, on the other hand, means that a subject indicated more relevant (TP) than irrelevant (FP) code smells. For code smell detection, a high number of missed code smells (false negatives) are worse than a high number of incorrect ones (false positives) because the false positives are revealed by the inevitable manual code inspection.

B. Joint Analysis for God Method Detection

Table V presents the overall results for the God Method detection. Rows in this table present three pieces of data: Recall (R), Precision (P), and the Time (T) in minutes used by subjects to complete their tasks. In total, 47 subjects had to identify God Method in the target system.

Data in this table shows that subjects in the traditional and hybrid groups achieved better results in terms of recall than subjects in the concern group. This result suggests that concern metrics when used in isolation do not offer appropriate means to detect God Method. Only one subject (S25) in the concern group scored more than 50% of recall. This performance is much worse than what the traditional and hybrid groups achieved; in average, they scored 65% and 55% of recall, respectively. In fact, this result is not a surprise since the God Method definition explicitly says about size and cohesion - attributes easily captured by traditional metrics.

However, we also observed that the concern metrics NCO, LOCC, NCC, and CDLOC were recurrently used by subjects of the hybrid group and they achieved high recall rates (Section IV.E). For instance, S42 scored 86% of recall and used LOC, PAR, and NCC. Similarly, S33 used LOC, PAR, NOO, and

TABLE V. Results for God Method Code Smell

Traditional Metrics

Subjects	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	**Avg**
Recall (%)	71%	57%	71%	71%	57%	71%	0%	86%	71%	71%	57%	86%	71%	71%	57%	71%	**65%**
Precision (%)	100%	67%	100%	100%	100%	100%	0%	100%	100%	100%	100%	100%	71%	100%	57%	71%	**85%**
Time (m)	11	13	9	13	14	7	10	6	10	15	7	15	7	8	12	7	**10**

Concern Metrics

Subjects	S17	S18	S19	S20	S21	S22	S23	S24	S25	S26	S27	S28	S29	S30	S31	S32	**Avg**
Recall (%)	14%	0%	0%	29%	0%	43%	29%	14%	57%	14%	43%	29%	0%	43%	29%	0%	**22%**
Precision (%)	100%	0%	0%	33%	0%	60%	50%	25%	100%	25%	100%	40%	0%	75%	0%	0%	**38%**
Time (m)	15	10	23	24	14	4	9	5	8	4	5	10	12	16	9	15	**11**

Hybrid Metrics

Subjects	S33	S34	S35	S36	S37	S38	S39	S40	S41	S42	S43	S44	S45	S46	S47	**Avg**
Recall (%)	71%	43%	57%	57%	86%	29%	43%	14%	57%	86%	57%	71%	43%	57%	57%	**55%**
Precision (%)	100%	38%	100%	100%	100%	50%	100%	33%	100%	100%	100%	100%	38%	100%	100%	**84%**
Time (m)	13	15	8	20	10	5	6	3	12	14	7	8	9	11	14	**10**

NCO to score 71% of recall. Therefore, we concluded that joint analysis of both concern and traditional metrics seems to also succeed in detecting this particular code smell.

C. *Analysis of the Concern Metrics*

The main goal of this paper is to evaluate the effectiveness of concern metrics to detect God Method. To fulfill this goal, we analyze in this section whether concern metrics succeeded in detecting God Method instances. As discussed in Section IV.B, we observed that four concern metrics (NCO, LOCC, NCC, and CDLOC) seem helpful when used together with traditional metrics to detect God Method.

The accuracy of the metric suite is largely dependent on the adequacy of each metric to quantify a property explicitly mentioned in the code smell definition. For instance, God Method is characterized by "a complex algorithm" [12] and "the realization of multiple responsibilities" [20]. While the first property is directly mapped to traditional metrics, the second one is better captured by concern metrics. This probably explains why many subjects using the hybrid metrics achieved good results for God Method detection.

D. *Analysis of Specific Metrics*

This section aims to identify a specific metric that may accurately detect the God Method. As explained in Section III.E, subjects reported the metrics they considered useful for God Method. Based on their answers, we analyze in this section the metrics that were considered useful to detect God Methods.

We analyzed the metrics that were considered useful to detect God Method by at least five subjects, as presented in Table VI. Rows of this table show the number of subjects that used each metric and the average of recall of these subjects. We also restricted our analyses to metrics with average of recall higher than 40%. The traditional metrics considered more useful were LOC and CYCLO. They were also those metrics that presented the highest recall rate: 64% and 63%, respectively. Another traditional metric with recall higher than 63% was PAR, aimed at quantifying the number of parameters in method signatures.

TABLE VI. METRICS CONSIDERED USEFUL FOR GOD METHOD

Metrics	LOC	CYCLO	NCO	LOCC	PAR	NCC	CDLOC
# Subjects who used	27	18	15	10	9	6	5
Average of recall (%)	64%	63%	53%	47%	63%	41%	46%

Four concern metrics were considered useful by 5 subjects or more: NCO, LOCC, NCC, and CDLOC. The metric NCO, aimed at computing the number of concerns in operations, was the concern metric that was used more often by the subjects. This concern metric was the one that presented the highest average recall, 53%, among subjects who used it. This seems to be an intuitive result as method is the locus of measurement for this metric and, in fact, the code smell is a structural problem at the method level. However, other three concern metrics at the class level were also considered useful: LOCC (recall of 47%), NCC (recall of 41%), and CDLOC (recall of 46%).

E. *Combined Use of Metrics*

In this section, we analyze possible combinations of metrics that when used together might be useful to detect specific God Method instances. To determine which metrics were used together to detect a code smell, we rely on an analysis of subjects who used the same metrics and scored higher in terms of recall.

In this God Method code smell analysis, we filtered subjects by considering only those who achieved more than 40% of recall. We observed some cases of metrics that were successfully used together. For instance, the combinations of LOC with CYCLO were used by 17 subjects that achieved more than 40% of recall. In addition, the concern metrics LOCC and NCO also succeed together. This combination was used by five subjects. Four subjects had the best performance in detecting God Method instances with 86% of recall each. They are (i) S2 and S37 using LOC and CYCLO; (ii) S12 using LOC, NOO, and PAR; (iii) S42 using LOC, PAR, and NCC.

F. *Analysis of Time Efficiency*

This section focuses on analyzing the time spent by subjects to detect God Method. Figure 1 shows data of recall (x-axis) and the time spent in minutes (y-axis) by each subject to detect this code smell. Each mark in this chart indicates a subject, but different symbols are used to distinguish the groups of metrics subjects worked with.

Data in Figure 1 help us to answer the question of whether a larger set of metrics can make the task of identifying God Method more time consuming (Section III.A). Note that, subjects in the hybrid group had to analyze a larger set of metrics, since they were provided with all 14 traditional and concern metrics (Section II.A). It is interesting to observe that, despite analyzing more data points, subjects in the hybrid group do not take longer to conclude their tasks. In fact, subjects in the concern group spent, in average, 11 minutes, against 10 minutes in both the hybrid and traditional groups.

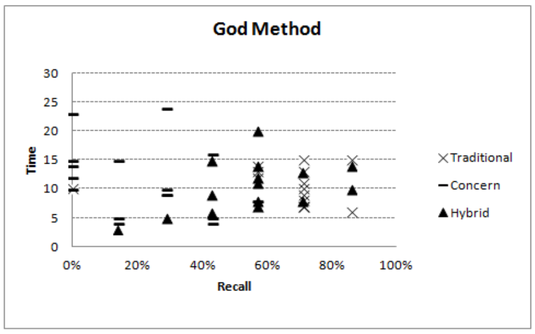

Fig. 1. Analysis of time efficiency for God Method

A careful analysis of Figure 1 also suggests that usually the longer the analysis, the better the results subjects achieved in terms of recall. This result can be confirmed by the fact that, most subjects who took more 10 minutes to analyze the data scored more than 50% of recall, regardless of the metrics they used. On the other hand, subjects using fewer metrics, i.e., in the concern metrics, were not time efficient. Therefore, we confirmed that the God Method detection requires careful analysis of many metrics, as indicated by the superiority of traditional and hybrid groups.

V. Study Constraints

The conclusions obtained here are restricted to the involved metrics, code smells, and the software system. These limitations are typical to exploratory studies like ours. We recognized these limitations, but our study fills the gap in the literature by reporting original analysis on whether the use of concern metrics is worthy for detecting the God Method code smell. Additionally, this paper describes the experimental framework that can be used in further replications of this study.

The accuracy of concern metrics depends on how accurate was the mapping (assignment) of concerns to code elements. Fortunately, we observed in a previous study [9] that, apart from Concern Diffusion over Lines of Code (CDLOC), the mapping process does not significantly impact on the concern metrics assessed in this paper. In addition, we relied on concern mappings produced by the original developers in order to mitigate this threat. Whether the concern mapping was fully correct or not, it just reflects how concern metrics would be used in practice.

VI. Conclusions and Future Works

The evaluation of software modularity is largely dependent on the availability of metrics that accurately detect code smells. Concern metrics are increasingly being used in empirical studies [4, 11, 13, 14]. Our study aims at examining the effectiveness of concern metrics to detect a code smell. Our general results revealed that some concern metrics might be useful to detect God Method.

We also investigated which specific metrics are more suitable to detect the analyzed code smell. In general, the results indicated that the accuracy of each metric suite is largely dependent on the adequacy of each metric to quantify a property explicitly mentioned in the smell definition. In particular, we observed that four concern metrics are able to help detecting God Method when used together with other traditional metrics.

This study represents a first stepping-stone towards the evaluation of concern metrics to detect code smells. We are currently working on strategies to detect this code smell (and others) based on the concern and traditional metrics we found. For future work, we plan to create and/or evaluate new metrics that better capture facets in others code smells.

Acknowledgements

This work was partially supported by Capes/Proex grants 2566/2012, FAPEMIG grants APQ-02376-11 and APQ-02532-12, and CNPq grant 485235/2011-0.

References

[1] Data of the Experiment with Concern Metrics, 2013: http://www.dcc.ufmg.br/~juliana.padilha/LaWasp2013/

[2] Carneiro, G. F. et al. “Identifying Code Smells with Multiple Concern Views”. In Proc. of the Brazilian Symposium on Software Engineering (SBES), 128-137, 2010.

[3] Chidamber, S. R. and Kemerer, C. F. “A Metrics Suite for Object Oriented Design”. Transactions on Software Engineering (TSE), 1994.

[4] Conejero, J. M. et al. “On the Relationship of Concern Metrics and Requirements Maintainability”, Inform. and Soft. Tech. (IST), 2011.

[5] Ducasse, S., Girba, T. and Kuhn, A. “Distribution Map”, In Proc. of the International Conf. on Software Maintance (ICSM), pp. 203-212, 2006.

[6] Eaddy, M. et al. “Do Crosscuting Concerns Cause Defects?”, IEEE Transactions on Software Engineering, pp. 497-515, 2008.

[7] Fenton, N. E. and Pfleeger, S. L. “Software Metrics: A Rigorous and Practical Approach”, 2nd ed. Thomson, 1996.

[8] Ferrari, F., et al.. “An Exploratory Study of Fault-Proneness in Evolving Aspect-Oriented Programs”. In Proc. of the International Conference on Software Engineering (ICSE), pp. 65-74, 2010.

[9] Figueiredo, E. , et al.. “On the Impact of Crosscutting Concern Projection on Code Measurement”, In Proc. of the International Conference on Aspect-Oriented Software Development (AOSD), 2011.

[10] Figueiredo, E. et al. "On the Maintainability of Aspect-Oriented Software: A Concern-Oriented Measurement Framework", In Proc. of the European Conf. on Soft. Maint. and Reengineering (CSMR), 2008.

[11] Figueiredo, E., et al. “Evolving Software Product Lines with Aspects: an Empirical Study on Design Stability”. In Proc. of the International Conference on Software Engineering (ICSE), pp. 261-270, 2008.

[12] Fowler, M. “Refactoring: Improving the Design of Existing Code”. Addison – Wesley, 1999.

[13] Garcia, A., et al. “Modularizing Design Patterns with Aspects: A Quantitative Study”. In Proc. of the International Conference on Aspect Oriented Software Development (AOSD), 2005.

[14] Greenwood, P. et al. “On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study”. In Proc. of the European Conference Object Oriented Programming (ECOOP), 2007.

[15] Kiczales, G., et al. “Aspect-Oriented Programming”. In Proc. of the European Conference O.O. Programming (ECOOP), pp. 220-242, 1997.

[16] Lanza, M. and Marinescu, R. “Object-Oriented Metrics in Practice”. Springer Verlag, 2006.

[17] Marinescu, R. “Detection Strategies: Metrics-Based Rules for Detecting Design Flaws”. In Proc. of the Intl Conf. on Soft. Maint. (ICSM), 2004.

[18] Nguyen, T., Nguyen, H., Nguyen, H., and Nguyen, T. “Aspect Recommendation for Evolving Software”. In Proc. of the International Conference on Software Engineering (ICSE), pp. 361-370, 2011.

[19] Parnas, D. L. “On the criteria to be used in decomposing systems into modules”. Communications of the ACM, 15(12), pp. 1053-1058, 1972.

[20] Riel, A. J. “Object-Oriented Design Heuristics”. Addison-Wesley Professional, 1996.

[21] Robillard, M., Murphy, G. “Representing Concerns in Source Code”, Trans. on Software Engineering and Methodology. 16(1), 2007.

[22] Sant’Anna, C., Garcia, A. and Lucena, C. “Evaluating the Efficacy of Concern-Driven Metrics: A Comparative Study”. In Proc. Workshop on Assessment of Contemporary Modularization Tech. (ACoM), 2008.

[23] Sant’Anna, C., et al. “On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework”. In Proc. of the Brazilian Symposium on Software Eng. (SBES), pp. 19-34, 2003.

[24] Silva, B. et al. “Concern-based Cohesion: Unveiling a Hidden Dimension of Cohesion Measurement”. In Proc. of the Int’l Conference on Program Comprehension (ICPC), 2012.