Prioritizing Software Anomalies with Software Metrics and Architecture Blueprints

A Controlled Experiment

Everton. Guimaraes, Alessandro Garcia
Informatics Department
PUC-Rio
Rio de Janeiro, Brazil
{eguimaraes, afgarcia}@inf.puc-rio.br

Eduardo Figueiredo
Department of Computer Science
Federal University of Minas Gerais
Belo Horizonte, Brazil
figueiredo@dcc.ufmg.brr

Yuanfang Cai
Department of Computer Science
Drexel University
Philadelphia, USA
yfcai@drexel.cs.edu

***Abstract*—According to recent studies, architecture degradation is to a large extent a consequence of the introduction of code anomalies as the system evolves. Many approaches have been proposed for detecting code anomalies, but none of them has been efficient on prioritizing code anomalies that represent real problems in the architecture design. In this sense, our work aims to investigate whether the prioritization of instances of three types of classical code anomalies, Divergent Change, God Class and Shotgun Surgery, can be improved when supported by architecture blueprints. These blueprints are informal models often available in software projects, and they are used to capture key architecture decisions. Moreover, we are also investigating what information may be useful in the design blueprints to help developers on prioritizing the most critical software anomalies. In many cases, developers indicated that it would be interesting the insertion of additional information on the blueprints in order to detect architecturally-relevant anomalies.**

***Index Terms*—Architectural Design, Software Metrics, Code Anomaly, Design Blueprint.**

I. INTRODUCTION

Architecture degradation [37] is a direct consequence of the progressive insertion of anomalies[1] in the source code as the system evolves [4][5][6]. The removal of an anomaly is of high priority in source code review if it is seen as harmful to the architecture design [3]. However, a high proportion of implementation anomalies, detected by source code analysis are not harmful to architecture design. Therefore, a crucial activity to ensure the software project longevity is to consistently distinguish, prioritize and remove architecturally-relevant code anomalies [4][6].

The use of metrics is the most popular way to identify candidates of architecturally-relevant code anomalies [14]. Developers can also define their own detection strategies, based on a particular combination of measures and thresholds, to automatically prioritize the removal of code anomalies [13]. However, recent studies [5][6] have revealed that previously-proposed detection strategies [13][14] fail to assist in the prioritization of architecturally-relevant code anomalies.

A prominent reason is that automatically collected measures often purely represent properties of the source code structure [5][6]. The measures alone are often agnostic to the architecture design structure. The architecture decomposition is often not explicit in the source code. As a result, developers tend to consider that all the measures and respective modules have the same relevance for them. On the other hand, architecture design models, available in software projects, are often of informal nature. They represent architecture blueprints [34] and are used for communication purposes only, making it unfeasible the collection of measures from them.

Given the omnipresence of architecture blueprints in many software projects, a natural question arises: to what extent their use would enhance the prioritization of architecturally-relevant code anomalies? The use of blueprints have been exploited and assessed in different software engineering activities, including process evaluation [36], model transformation optimization [38], and test coverage analysis [35]. However, there is no empirical knowledge on how the detection of architecturally-relevant code anomalies would be (or not) improved with explicit reasoning about architecture blueprints.

In this sense, this work aims to investigate to what extent the use of design blueprints, typically available in software projects for communication purposes, would enhance the prioritization of critical code anomalies. Furthermore, we are investigating what information is useful to be showed in the design blueprints. In many cases, developers indicated that would be interesting the insertion of additional information (i.e. number of attributes/methods and dependency strength) on the blueprint in order to identify architecturally-relevant software anomalies. This information would complement the information provided by the metrics based on source-code and thus, help developers to better detect the anomalies. This controlled experiment allows us to answer questions like: (i) how design blueprints generated by developers can help to detect software anomalies instead of using an strategy based on

[1] Implementation anomalies are also well-known as code smells [9]

source code information; and (ii) which information represented in the design blueprints may be useful to help on the anomaly prioritization process.

II. Background and Related Work

This section describes the main concepts involved in our study. Section A introduces the concept of software anomalies. Section B presents the metrics used in our study. Section C presents the design blueprints used in conjunction with software metrics to detect software anomalies.

A. Software Anomalies

Software anomaly is a situation that suggests a potential problem on software structure [9]. To decide whether the anomaly is critical to the architectural design or not requires deeper investigation. While some types of anomalies can be observed only looking at the source code, others are observed in other software artifacts (e.g., in architecture blueprints). It is important to investigate presence of anomalies in different abstraction levels since developers have to identify, for instance, which code anomalies may impact on software design during its evolution. Research work has highlighted the impact of code anomalies on undesirable modifications in the source code [4][18][25]. For instance, recent studies revealed that the source code structure affected by anomalies is prone to change in the software evolution [18][25]. It is also known that code anomalies favor the occurrence of faults [4].

In this study, we investigate three classic anomalies, which are often related to architectural problems [4][5][6]: God Class, Divergent Change, and Shotgun Surgery. *God Class* represents a class that has grown beyond all logic to become the class that does almost everything in the system [24]. A god class implements too many responsibilities. *Divergent Change* can be observed when a class is commonly changed in different ways and for different reasons [9]. *Shotgun Surgery* can be identified when a class change leads to a lot of small changes in many different classes [9]. Shotgun Surgery is somehow the opposite of Divergent Change. We also decided to focus on these three anomalies because they recurrently occur in software systems [17], and previous studies have related them to poor modularization of concerns detected by design blueprints [13].

A. Software Metrics for Anomalies Detection

Software metrics are an important mechanism of software engineering to evaluate the quality of software solutions. For instance, the detection of code anomalies by means of software metrics has been investigated in many works [13][14]. Table I summarizes the traditional and concern metrics used for detecting software anomalies in our experiment. We selected only a subset of the most commonly used metrics because a big number of metrics could make its analysis harder and bring redundant measurements. The selected traditional metrics have been used in previous work and have proved to help developers on maintenance tasks, such as code anomalies detection [13].

Moreover, concern metrics are mainly focused on the quantification of modularity properties, such as the degree of scattering and tangling of the system concerns. Concern scattering is defined as the degree in which a concern spreads over the design elements, while concern tangling represents the degree in which a concern is mixed up with other concerns on the system modules.

TABLE I. Description of Traditional and Concern-based Metrics

Metric	Brief Definition
Number of Attributes (NOA)	It counts the number of attributes in a class.
Number of Methods (NOM)	It counts the number of methods in a class.
Weighted Methods per Class (WMC)	It counts the number of methods and their parameters in a class.
Lines of Code (LOC)	It counts the total number of lines of code.
Lack of Cohesion in Methods (LCOM)	It is measured by the number of pairs of methods that do not access attributes in common by the number of pairs of methods that access attributes in common.
Coupling Between Objects (CBO)	It counts the number of classes in which a given class either calls its methods or accesses its attributes.
Number of Concerns per Component (NCC)	It counts the number of concerns in each class
Concern Diffusion over Components (CDC)	It counts the number of classes affected by the concern implementation.
Concern Diffusion over Operations (CDO)	It counts the number of methods affected by the concern implementation.
Concern Diffusion in LOC (CDLOC)	It counts the number of transition-points for each concern through the lines of code.

B. Architecture Blueprints

The concept of blueprints is used to describe each of the architectural views proposed in the "4+1 View Model" [34], which describes software architecture using five concurrent views. In this work, we are using both the development and logical views, which are represented by the component and class diagrams, respectively. For the best of our knowledge, the *development view* focuses on the organization of the actual software modules, while the *logical view* is concerned with the functionality provided to end-users.

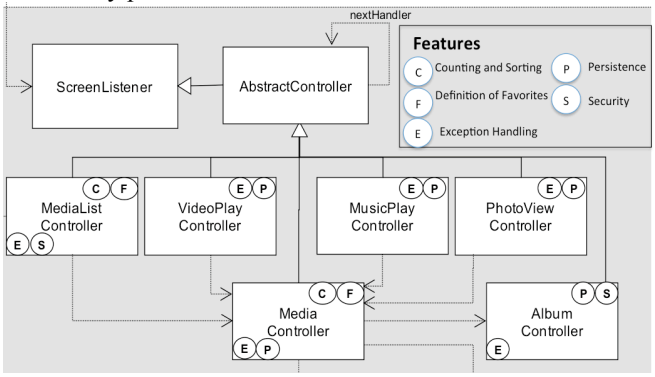

Fig. 1 - Partial View of Mobile Media Architecture

Figure 1 depicts a component blueprint representing a partial view of Mobile Media architecture [39]. This system is discussed in Section III.B. The rectangle represents the main classes of the controller layer. In this component blueprint, each class is decorated with small circles to represents the concerns it implements. For instance, VideoPlayController implements the Exception Handling (e) and Persistence (P)

concerns. In addition to the component blueprint, we have provided a class blueprint. That is, a UML class diagram with all Mobile Media classes and their relationships.

We evaluate the usefulness of software blueprints in this study as a way of detecting software anomalies because software blueprints have also been successfully used in different domains. For instance, software blueprints have been used for (i) implementing visual aid to assess test coverage [35] and (ii) implementing a visual approach for software process model evaluation based on architectural views [36]. These blueprints were used to find errors and flaws in the specification and improvement phases of the software process.

III. Study Methodology

This section introduces the methodology adopted in our study. Section A describes the study hypotheses and Section B introduces the used application. Section C presents the code smells reference list. Section D explains the experimental procedures we follow.

A. Research Question and Hypotheses

Our study goal is to investigate whether design blueprints support the use of software metrics on detecting software anomalies. We define the following research question (*RQ*). We derived the *null* and *alternative* hypotheses from our research question, as presented in Table II. We investigate the values of Precision and Recall (Section IV) when subjects were provided with software metrics (*m*) and when they were provided with software metrics and design blueprints (*md*).

RQ: How can architecture blueprints help the prioritization of relevant code anomalies?

TABLE II. Summary of hypotheses

Hypothesis	Description
$H_{1.0}$	Precision (md) ≤ Precision (m)
$H_{1.1}$	Precision (md) > Precision (m)
$H_{1.0}$	Recall (md) ≤ Recall (m)
$H_{1.1}$	Recall (md) > Recall (m)

The first hypothesis H_1 is concerned with the impact of the use of metrics and design blueprints on detecting software anomalies. The null hypothesis $H_{1.0}$ states that the use of software design as additional artifact to detect anomalies do not provide any enhancement on the detection process. On the other hand, the alternative hypothesis $H_{1.1}$ states that the precision on the detection of software anomalies was higher when developers are provided with design blueprints.

The second hypothesis H_2 is concerned with the impact of the use of design blueprints on the recall measures. Similarly to the first hypothesis, the null hypothesis $H_{2.0}$ states that the use of design blueprints to improve the anomaly detection process do not impact on the recall measures. On the other hand, the alternative hypothesis $H_{2.1}$ states that the recall measures were lower when developers are provided with metrics plus design blueprints.

B. Target Application

We selected the Mobile Media system to be used in this study. Mobile Media is a software product line that provides support for manipulation of photos, music and videos on mobile devices [39]. The main reasons we selected Mobile Media are: (i) the original developers produced the architectural design and architectural blueprints - blueprints are the artifacts used to reason about changes requests and derive new products during the system evolution; (ii) different types of change were realized in each release, including refinements of the architecture style employed; and (iii) the system has been successfully used in other studies involving empirical evaluation [27]. More details about the Mobile Media characteristics can be found elsewhere [34].

C. Code Smells Reference List

In order to conduct our experiment, we performed a systematic analysis of Mobile Media aiming to identify classes that are affected by relevant anomalies. Moreover, we relied on two experts in the target system to build the reference list for each code smell being analyzed in this study. The experts were involved in the development, maintenance or assessment of the target application. Table III shows reference list of anomalies in the Mobile media after a cross-discussion take place among experts and ourselves to reach a joint decision.

TABLE III. Reference List of Code Anomalies

Code Anomaly	Classes
God Class	MediaAccessor, MediaController
Shotgun Surgery	ControllerInterface, MediaAssessor, ScreenSingleton
Divergent Changes	MediaListController, ImageMediaAccessor, MediaAssessor, MediaController,

D. Experimental Procedures

This section describes the experimental procedures defined in our controlled experiment. Firstly, a 20-minute training session was performed in order to introduce subjects with the main concepts of this study, such as the metrics, the anomalies, and blueprints. We then organized subjects into two groups: those who have not used design blueprints (NBP) and those who used design blueprints (BP). The NBP group was provided only with a set of software metrics (Section II.B). The BP group was provided with design blueprints and metrics. Moreover, both groups of subjects received a document with a partial view of Mobile Media, a brief explanation of system architectural design and a description of the concerns involved in the target application. Finally, subjects were asked to reason about the artifacts provided to them in order to identify which classes could be candidates to present the anomalies.

IV. Detecting Software Anomalies

This section presents the experiment results. In order to compare the efficiency of anomalies detection with and without blueprints, we used Precision and Recall measures, which leverage three other metrics: (i) True Positives (TP) – the number of correctly identified anomalies; (ii) False Positives (FP) – the number of wrongly identified anomalies; and (iii) False Negatives (FN) – the number of code anomalies that were missed. Those three measures are collected based on

the information provided by the subjects. The Precision and Recall were calculated as defined in Table IV.

TABLE IV. PRECISION AND RECALL

Precision	Recall
Precision = TP/(TP+FP)	Recall = TP/(TP+FN)

Recall (R) measures the fraction of the anomalies detected by the subjects to the total number of anomalies, as listed in the reference list. Precision (P) measures the ratio of correctly detected code smells by the total number of anomalies identified by subjects. A high recall implies that a subject identified most of the relevant anomalies. On the other hand, a high precision implies that the subject identified more relevant anomalies than the irrelevant ones. We focus on recall because it is important not to miss many anomalies.

Table V shows the descriptive statistics of precision and recall measures for BP group. The table is organized in 5 columns: (i) Measures – it describes whether the measures either for precision or recall; (ii) Anomaly – it describes the anomaly that is been assessed. We used an acronym for each anomaly: Divergent Change (DC), God Class (GC) and Shotgun Surgery (SS); (iii) N – it represents the number of pairs of subjects that were assigned to detect a specific anomaly; (IV) Mean – It represents the average either for precision or recall reached for a group of subjects; and (V) S.D. – It represents the standard deviation either for the precision or recall measures.

TABLE V. DESCRIPTIVE STATISTIC FOR PRECISION AND RECALL MEASURES

Measures	Anomaly	N		Mean (%)		S.D. (%)	
		BP	NBP	BP	NBP	BP	NBP
Precision	DC	10	24	47,9	43,62	26.7	27.1
	GC	14	10	50.9	66.8	25.5	23.6
	SS	11	20	25,8	21.5	5.4	19.8
Recall	DC	10	24	44.5	39.17	16.7	28.4
	GC	14	10	82.1	73.3	24.9	30.8
	SS	11	20	33.0	21.31	0	24.3

We can observe that subjects with blueprints obtained higher precision for Divergent Changes and Shotgun Surgery detection, but not for the detection of God Class anomaly. The precision for God Class detection was around 50.9% for the BP group, versus 66.8% for the NBP group, with a difference of around 15%. It is also interesting to observe that the BP group also revealed higher standard deviation for God Class detection, which implies that the data are spread over a broader range of values. We also observe that the BP group obtained higher recall measures for all the three anomalies. The recall measure differences between the two groups were 5.3%, 8.8% and 11.7%, for the anomalies Divergent Changes, God Class and Shotgun Surgery respectively. Moreover, the standard deviation measures were higher in the NBP group for all the cases. It is even worse for the case of the Shotgun Surgery anomaly, where the S.D. was 24.3% for NBP group, while in BP group the S.D equals to 0. It means that the number of anomalies detected by the BP group members is highly consistent. More detailed analysis follows.

A. Design Blueprints and Precision

Our first objective is to investigate whether the use of design blueprints may improve the accuracy of anomaly detection. We first examine and compare the precision measures. Thus, we observe that the precision measure for the BP group has increased for two of the three anomalies. For instance, the precisions measures for Shotgun Surgery detection from both groups are lower than 30%, meaning that it is very hard to detect this type of anomaly. The precision increased about 4% when the subjects are provided with the design blueprints. Similarly, the precision measures for the Divergent Change increased about 4.3% with the help of blueprints.

However, for God Class detection, we observed that the precision was not improved in the BP group. The difference between the precision measures of the two groups for this anomaly was around 16%, in favor of NPB group. The God Class anomaly is more intuitive to detect than the other two anomalies. As a result, in some cases the subjects might not have followed the inspection process correctly when detecting God Classes. It can lead to a higher number of False Positives, for instance, because of the misinterpretation of metrics values. The higher the number of False Positives, the lower is the precision measures observed.

In order to test hypothesis $H_{1.0}$, we have performed the Wilcoxon signed-rank test. We choose this test because the dataset for the two groups have different sizes. In this test, we chose a significance level of 0.05 as the threshold to confirm or refute the hypothesis. Furthermore, in order to draw valid conclusions, it is important to confirm that the outliers caused by extraordinary exception have been properly removed. For instance, we observed two cases in the NBP group where the subjects have not identified any anomaly correctly. That is, for these cases the number of TP = 0, and due the fact that the number of TP composes the precision and recall measures, we decided to remove these two samples. Table VI shows the mean value of precision for each group and the calculated p-value.

TABLE VI. PRECISION MEASURES BY GROUP

Measure	Group	N	Mean (%)	Calc. P-value
Precision	BP	35	46.2	0.1
	NBP	55	39.7	

We observe that although the NPB group had a higher number of subjects working on the detection of the three anomalies, the BP group presented better average precision measures. However, the application of Wilcoxon signed-rank test results in a p-value of 0.1. Thus, the null hypothesis $H_{1.0}$ cannot be rejected. In other words, the alternative hypothesis $H_{1.1}$ that the difference in the precision measures is higher when subjects are provided with design blueprints cannot be confirmed using the significance threshold of 0.05. Thus, we cannot draw valid statistical conclusions that using design blueprints can improve anomaly detection by analyzing only

the precision measures. It means that more experiments are still needed in order to achieve statistically significant results.

B. Design Blueprints and Recall

In this section, we will analyze the impact of using design blueprints on recall measures (see Tables V and VII). In our study, the recall indicates the percentage of the number of anomalies detected by the subjects to the total number of anomalies in the oracle anomaly list. In the data analysis, we, observed that the recall measures for Shotgun Surgery anomaly was higher than 30% when the subjects were provided with design blueprints, 12% higher than the recall measures from the NBP group. Similarly, the recall measures for the Divergent Change and God Class increased about 5.3% and 8.8%, respectively, when the subjects were provided with blueprints.

In order to test the hypothesis $H_{2.0}$, we similarly applied the Wilcoxon signed-rank test with a significance level of 0.05. From Table VII, we observe that, in average, the recall measures were higher for the BP group. That is, when subjects were provided with design blueprints, the average recall measures increased by 21%. It shows that the use of design blueprints improved the effectiveness of anomaly detection by decreasing the number of False Negatives. The lower the number of False Negatives, the higher is the Recall measures. Moreover, we have applied the statistical test in order to assess whether there is a statistically significant difference on the recall measures.

TABLE VII. RECALL MEASURES BY GROUP

Measure	Group	N	Mean	Calc. P-value
Recall	A	35	62.3	0.001
	B	54	41.3	

The test of hypothesis H_2 follows the same procedures performed to test the hypothesis H_1. The results showed a calculated p-value < 0.001. Therefore, the null hypothesis that the use of design blueprints has no impact on the recall measure can be rejected. Thus, we can conclude that in general, there is a difference between the recall measures when subjects were provided with design blueprints and when they were not. In other words, we can conclude that using design blueprints could improve anomaly detection process in that fewer anomalies will be missed.

C. Usefulness of Design Blueprints

Besides the tasks of detecting software anomalies, we asked the subjects to indicate whether they judge the design blueprint as being useful on the detection process (see Figure 2). About 71.4% of subjects judged that the design blueprints provided in the experiment as being useful in the detection process. Only 28.6% claimed that the design blueprints have not been useful, so they used only the set of metrics provided in the experiment to detect the design anomalies.

For the subjects that indicated the design blueprints have been an useful artifact on the detection process, we also asked them to point out what kind of information would be useful to be added in the blueprint. Most part of the subjects said that the information provided on the design blueprints were complete and no additional information is needed. That is, for those subjects the information presented on design blueprints was sufficient to help with the detection process. Moreover, some subjects suggested that the blueprints should reveal classes with a high number of attributes and methods, such classes could be considered as a good candidate for the anomaly *God Class*. Base on this information, the subjects can pre-select the classes that might have a specific anomaly. Their suspicion can then be confirmed or rejected through the evaluation of metrics.

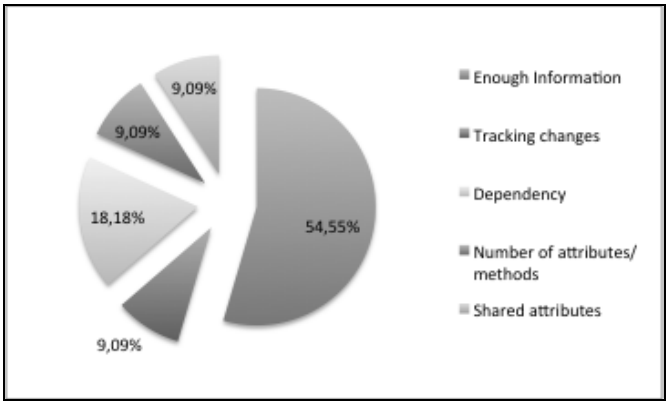

Fig. 2 - Information Suggested by Subjects

Another sample suggestion was that the information about which classes have a high number of dependencies could be useful if included in the design blueprint. In this case, the subjects can detect classes that have the anomalies *Divergent Changes* or *Shotgun Surgery*. Once the classes are pre-selected, the metrics can be used to confirm if a given class has an anomaly. In summary, design blueprints were considered to be useful because the subjects can initially identify candidate classes that potentially have an anomaly.

V. FINAL REMARKS AND FUTURE WORK

In this work, we performed a controlled experiment, in order to investigate how the process of prioritizing architecture-relevant anomalies can be improved, when guided by design blueprints. The subjects were organized into 2 groups. One group was provided only with code artifacts, such as metrics e source-code information, while the other group was also provided with design blueprints. For the second group, the design blueprints should be the main artifacts to be analyzed in on the prioritization process.

Our initial findings show that to some extent, the design blueprints has improved both precision and recall measures. For instance, we could observe that the precision measures were higher for Divergent Change and Shotgun Surgery. However, in the case of God Class anomaly the results were not expressive and the design blueprints have not improved the detection process. In general, for the cases where the use of design blueprints have improved the precision measures, the results were not higher than 20%. According to the feedback provided by the subjects, we observed that some of them have not followed correctly the inspection sequence defined in our experiment because they considered the anomaly God Class as being more intuitive to detect. This may have lead to the detection of more False Positives, and as consequence, had

negative impact on precision measures. On the other hand, when we observe the recall measures the results were different. The recall has increased for all the three anomalies. Thus, as the recall measures indicate the sensitivity of subjects on detecting real positive cases that are correctly predicted as positive, we could observe a lower number of False Negatives. A lower number of False Negatives implies in a higher recall. Independently whether the subjects have used correctly the design blueprints, we observed that the use of this artifact has somehow improved occurrence of False Positives and False Negatives.

In summary, when we tested the hypothesis $H_{1.0}$ could not be rejected because the calculated p-value was higher than the acceptable significance level ($\alpha = 0.05$). Therefore, even though we observed that the design blueprints can somehow improve precision measures, the statistical tests indicate cannot be considered as being statically significant. On the other hand, when testing the hypothesis $H_{2.0}$ we observed that the recall measures were affected by the used of design blueprints on the detection process. Since the calculated p-value was lower than the significance used on the hypothesis test, the hypothesis H2.0 can be rejected. Finally, it is important to highlight that the experiment was conducted with undergrad and graduate students in two different universities. Since, this a not a limitation of our study since it is a first investigation on how the detection of software anomalies can be improved, when guided by design blueprints. As a future work, we intend to invite experienced developers to participate of our experiment. The controlled experiment with experience developers may provide more interesting results regarding the usefulness of design blueprints on the detection process. Furthermore, we also want to investigate more software anomalies that have been considered as being relevant for software design.

ACKNOWLEDGEMENTS

This work has received full or partial funding from the following agencies and projects: for all the authors PROCAD-NF (grant 720/2010); for Alessandro - FAPERJ (distinguished scientist grant E- 26/102.211/2009), DANSis project (grant E-26/111.152/2011), CNPq (productivity grant 305526/2009- 0), PUC-Rio (productivity grant), Universal Project (grant 485348/2011-0), CAPES: international collaboration scheme (grants 5688-09, 021/2011).

.

REFERENCES

[1] F. J. Pérez. Refactoring Planning for Design Smell Correction in Object-Oriented Software. Escuela Técnica Superior de Ingeniería Informática, Universidad de Valladolid, PhD Thesis, Spain, 2011.

[2] M. Salehie, S. Li and L. Tahvildari. A Metric-Based heuristic Framework to Detect Object-Oriented Design Flaws. In Prof. Of 14th ICPC, 2006.

[3] J. Garcia, D. Popescu, G. Edwards and N. Medvidovic, Identifying Architectural Bad Smells. In Proc. of 13th CSMR, March 2009.

[4] I. Macia, A. Garcia and A. von Staa. An Exploratory Study of Code Smells in Evolving Aspect-Oriented Systems. In Proc. of 10th AOSD, pp. 203-214, Recife, March 2011.

[5] I. Macia, J. Garcia, D. Popescu, A. Garcia, N. Medvidovic and A. von Staa. Are Automatically-Detected Code Anomalies Relevant to Architectural Modularity? . In Proc. of 11th AOSD, pp. 167-178, USA, March 2012.

[6] I. Macia, R. Arcoverde, A. Garcia, C. Chavez and A. von Staa. On the Relevance of Code Anomalies for Identifying Architecture Degradation Symptoms. In Proc. of 16th CSMR, Szeged, Hungary, March 2012.

[7] N. Moha, Y. Guéhéneuc, L. Duchien and A. F. Meur. DECOR: a Method for the Specification and Detection of Code and Design Smells. In Proc. of IEEE Transactions on Software Engineering, Vol. 36, Issue 1, pp. 20-36, January/February 2010.

[8] J. Ratzinger, M. Fisher and H. Gall. Improving Evolvability through Refactorings. In Proc. of the Int'l Workshop on Mining Software Repositories. Vol. 30, pp. 1-5, July 2005.

[9] M. Fowler and K. Beck. Refactoring: Improving the Design of Existing Code, Addison-Wesley, USA 1999.

[10] N. Tsantalis, T. Chaikalis and A. Chatzigeorgiou. JDeodorant: Identification and Removal of Type-Checking Bad Smells. In Proc. of 12th CSMR, pp. 329-331, April 2008.

[11] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object-Oriented Design. IEEE Transaction of Software Engineering. Pp.476-493, 1993.

[12] B. Henderson. Object-Oriented Metrics: Measures of Complexity. Prentice-Hall Inc., New Jersey, USA 1996.

[13] R. Marinescu. Detection Strategies: Metrics-Based Rules for Detecting Design Flaws. In Proc. of IEEE ICSM, pp. 350-359, September 2004.

[14] M. Lanza and R. Marinescu. Object-Oriented Metrics in Practice – Using Software Metrics to Characterize, Evaluate and Improve the Design of Object-Oriented Systems, Springer-Verlag, 2006.

[15] S. Demeyer, S. Ducasse and M. Lanza. A Hybrid Reverse Engineering Approach Combining Metrics and Program Visualization. In Proc. of 6th WCRE, pp. 175-186, 1999.

[16] G. Carneiro, M. Carneiro, L. Mara, C. Sant'Anna, A. Garcia, M. G. Mendonça. Identifying Code Smells with Multiple Concern Views. In Proc. of 1st CBSOFT, Salvador, Brazil 2010.

[17] S. Olbrich, D. S. Cruzes, V. Basili and N. Zazworka. The Evolution and Impact of Code Smells: a Case Study of Two Open Source Systems. In Proc. of 3rd Int'l Symposium on ESEM, pp. 390-400, Florida, USA, September 2009.

[18] F. Khomh, M. Penta and Y. Guéhéneuc. An Exploratory Study of the Impact of Code Smells on Software Change-Proneness. In Proc. of 16th WCRE, pp. 75-84, 2009.

[19] F. Fenton and S. Pfleeger. Software Metrics: a Rigorous and Practical Approach, Course Technology (Revised), 1998.

[20] A. Trifu. Towards Automated Restructuring of Object-Oriented Systems. PhD Thesis, Universität Karlsruhe, 2008.

[21] W. J. Brown, R. C. Malveau, H. W. McCormick and T. J. Mowbray. AntiPatterns: Refactoring Software, Architectures and Projects in Crisis, John Wiley and Sons, March 1998.

[22] B. F. Webster. Pitfalls of Object-Oriented Development. M & T Books, February 2005.

[23] A. Tiberghien, N. Moha, T. Mens and K. Mens. Répertoire des Défauts de Conception, Technical Report 1303, University of Montreal, 2007.

[24] A. Riel. Object-Oriented Design Heuristics. Addison-Wesley Longman Publishing Co. Boston, MA, April 1996.

[25] M. Mantyla, J. Vanhanen and C. Lassenius. A Taxonomy and an Initial Empirical Study of bad Smells in Code. In Proc. of 19th ICSM, Amsterdam, Netherland, September 2003.

[26] W. Li and R. Shatnawi. An Empirical Study of Bad Smells and Class Error Probability in the Post-release Object-Oriented System Evolution. Journal of Systems and Software, Vol. 80, pp. 1120-1128, July 2007.

[27] E. Figueiredo, C. Sant'Anna, A. Garcia, T. Bartolomei, W. Cazzola and A. Marchetto. On the Maintainability of Aspect-Oriented Software: a Concern-Oriented Measurement Framework. In Proc. of 12th CSMR, pp. 183-192, 2008.

[28] E. Figueiredo, A. Garcia, M. Maia, C. Nunes, and J. Whittle. On the Impact of Crosscutting Concern Projection on Code Measurement. In Proc. Of 10th AOSD, pp. 81-92, 2011.

[29] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, L. Videira, J. Loingtier and J. Irwin. Aspect-Oriented Programming. In Proc. of 22nd OOPSLA, Springer, LNCS 1241, pp. 220-242, Finland 2007.

[30] M. Robillard and G. Murphy. Representing Concerns in Source Code. In Proc. of TOSEM, Vol. 16, February 2007.

[31] S. Soares, E. Laureano and P. Borba. Implementing Distribution and Persistence Aspects with AspectJ. In Proc. of 17th ACM OOPSLA, vol. 37, issue 11, pp. 174-190, November 2002.

[32] M. D'Ambros, A. Bacchelli and M. Lanza. On the Impact of Design Flaws on Software Defects. In Proc. of 10th Int'l Conf. Software Quality, pp. 23-31, 2010.

[33] I. Deligiannis, I. Stamelos, L. Angelis, M. Roumeliotis and M. Shepperd. A Controlled Experiment Investigation of an Object-Oriented Design Heuristics for Maintainability. Journal of Systems and Software, Vol. 72, Issue 2, pp. 129-143, 2004.

[34] P. B. Kutchen. Architectural Blueprint: The "4+1" View Model of Architecture. In Proc. of IEEE Software, Vol. 12, Issue 6, pp. 42-50, 1995.

[35] V. A. Araya. Test Blueprint: An Effective Visual Support to Test Coverage. In Proc. of 33rd ICSE, pp. 1140-1145, 2011.

[36] J. A. Alegría, A. Lagos, A. Bergel and M. C. Bastarrica. Process Model Blueprints. In Proc. of ICSP, Springer Verlag, pp. 273-284, 2010.

[37] L. Hochstein and M. Lindvall. Combating Architectural degenerations: A Survey. Information and Software Technology, Vol. 47, Issue 10, pp. 643-656, July 2005.

[38] C. Jeanneret, M. Glinz and B. Baudry. Estimating Footprints of Model Operations. In Proc. of 33rd ICSE, pp. 601-610, 2011.

[39] E. Figueiredo, et al. Evolving Software Product Lines with Aspects: an Empirical Study on Design Stability, Proc. of the Int. Conf. on Software Engineering (ICSE), 261-270, 2008.