STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LX**, Number 2, 2015

METRICS-BASED REFACTORING STRATEGY AND IMPACT ON SOFTWARE QUALITY

SIMONA MOTOGNA, CAMELIA ŞERBAN, AND ANDREEA VESCAN

Abstract. Nowadays, software systems become very large and complex applications. They are the result of a process evolution, undergoing frequent modifications that affect their design quality. Therefore, repeated assessment of the systems design should be made throughout the development lifecycle. This activity is time-consuming, because of design complexity and therefore, methods and techniques being needed to evaluate the system design in an automatic manner. Software metrics are an alternative solution, a means for quantifying those aspects considered important for the assessment.

In this article we propose a case-study for object oriented design (OOD) assessment using software metrics. Our goal is to identify those design entities affected by “God Class” design flaw, and to identify possible refactorings that could improve the system design and to evaluate the impact of the applied refactoring on software quality.

1. Introduction

Over an extended period of time software systems are often subject to a process of evolution becoming very large and complex applications. They undergo repeated modifications in order to satisfy any requirement regarding a business change. The result is that the code deviates from its original design and the system becomes unmanageable. A minor change in one of its parts may have unpredictable effects in completely other parts [1]. To avoid such risk a high quality design should be preserved throughout the system life cycle. This can be achieved by repeatedly assessing the system design, aiming to identify in due course those design entities that do not comply with the rules, principles and practices of a good design, and suggesting possible refactorings or improvements to be performed.

Received by the editors: September 27, 2015.

2010 *Mathematics Subject Classification.* 68N30, 68T37.

1998 *CR Categories and Descriptors.* D.2.8. [**Software Engineering**]: Metric – *Product Metrics*; D.2.8. [**Software Engineering**]: Refactoring – *Extract Method*; D.1.5. [**Pattern recognition**]: Clustering – *Fuzzy Clustering*.

Key words and phrases. Software metrics, refactoring, fuzzy clustering.

Due to the complexity of OOD, its assessment becomes a time-consuming activity. Consequently, methods and techniques are needed in order to evaluate the system design in an automatic manner. Software metrics are an alternative solution, being a means for quantifying those aspects considered important for the assessment.

Our previous work [2] was focused on developing a methodology for quantitative evaluation of OOD. The proposed methodology is based on static analysis of the source code and is described by a framework of four abstraction layers. To complete the interpretation of the obtained measurement results, a new metric, named Design Flaw Entropy (DFE), which measures the distribution of a specified design flaw among the analyzed design entities, was also proposed [7].

Starting from the above mentioned our previous proposed methodology for OOD, the current article adds a new step in the process of OOD assessment. More precisely, after the interpretation of the obtained measurements results which has the goal of identifying "suspect" design entities, we suggest possible refactorings and we study their impact regarding the design improvement. In other words, at this step we apply some refactorings and we repeat the evaluation for the obtained design, comparatively studying the two sets of metrics values.

The paper is organized as follows. Section 2 briefly describes the process of OOD assessment, whereas in Section 3 we present a case-study in order to validate our approach for design assessment. Section 4 summarizes the contributions of this work and outlines directions for further research.

2. Object oriented design assessment process

The steps needed to be perform in order to apply our previous proposed methodology for OOD assessment are described in what follows.

2.1. **Setting the assessment objectives.** The first step of OOD assessment process is to establish the assessment objectives. In this study we aim to identify those design entities affected by "God Class" [6] design flaw. Consequently, the assessed entities, are the set of classes from the analyzed design system.

An instance of God Class does most of the operation tasks, leaving only minor details to a series of trivial classes; it also uses the data from other classes. Briefly, *God Class* design flaw refers to those classes "which tend to centralize the intelligence of the system" [1]. As a consequence, the principle of manageable complexity is violated, as god classes tend to capture more than one abstraction. Another shortcoming of these pathological classes is their tendency towards non-cohesion. If we consider the quality attributes, god-classes also have a negative impact on the reusability and understandability of that part of the system they belong to. To detect a God Class, Salehie et al. [5] have related this design flaw to a set of three heuristics [4]:

- distribute system intelligence horizontally as uniformly as possible;
- beware of classes with much non-communicative behavior;
- beware of classes that access directly data from other classes.

Therefore, each class design entity, will be evaluated in respect with the above mentioned heuristics.

2.2. **Metrics identification.** Having identified some heuristics which are correlated with "God Class" design flaw, our goal is to find for each heuristic relevant metrics. Looking at the above heuristics the conclusion drawn may be:

- the first rule suggests that it should be a uniform distribution of intelligence among classes, so it refers to high class complexity;
- the second rule refers to the level of intra-class communication; i.e. to weak cohesion of classes;
- the third rule refers to classes that use a lot of data from other classes.

Thus, the selected heuristics are then related with the following metrics:

- Cyclomatic complexity [11] is a measure of a module control flow complexity based on graph theory. A control flow graph describes the logic structure a software module. Each flow graph consists of nodes and edges. The nodes represent computational statements or expressions, and the edges represent the transfer of control between nodes [12]. Cyclomatic complexity is defined for each module to be e n + 2, where e are the number of edges and n are the number of nodes in the control flow graph.

  Impact of CC metric value on software quality. A high value for the cyclomatic complexity metric indicates a low quality code which might involve difficulties in testing and maintaining.

  The CC metric is defined on methods. Adapted to the object oriented world, this metric [10] is also defined for classes and structures as the sum of its methods CC.
- Lack of Cohesion Of Methods (LCOM) [8]. LCOM is defined by the difference between the number of method pairs using common instance variables and the number of method pairs that do not use any common variables.

  Impact of CC metric value on software quality. The single responsibility principle states that a class should not have more than one reason to change. Such a class is said to be cohesive. A high LCOM value generally pinpoints a poorly cohesive class. Looking into LCOM metric, Henderson-Sellers [9] points out the large number of dissimilar situations with 0 value of LCOM.

- Efferent Coupling at type level (Ce). The Efferent Coupling for a particular type is the number of types it directly depends on. Notice that types declared in third-party assemblies are taken into account.

  Impact of CC metric value on software quality. Types where Ce ¿ 50 are types that depends on too many other types. They are complex and have more than one responsibility. They are good candidate for refactoring.

Thus, at this step of the assessment process, each class can be viewed as a vector with the corresponding values of the LCOM, CC, Ce metrics.

Analyzing the definitions of these metrics and taking into account the above mentioned principles and heuristics related with “God Class”, we can conclude that a possible God Class suspect will have high values for the LCOM, Ce si CC metrics.

2.3. **Fuzzy based God Class Detection and DFE metric.** Having computing the metrics values, for each class design entity, the next step is to identify the list of “suspect”, i.e those classes affected by “God Class” design flaw. In order to obtain these entities we use fuzzy clustering analysis. This method overcome the limitations of the existing approaches which use thresholds values for metrics, thus an entity (class) will be place in more than one group (cluster), having different membership degree. Each cluster of the obtained partition is analyzed in detail, in order to decide if it contains “suspect” classes.

A metric proposed in our previous work [7], Design Flaw Entropy, provides an in-depth analysis regarding the distribution of the analyzed design flaw (the degree of its spread) into the system.

2.4. **Proposed Refactorings and their impact.** At the step of the assessment process, we aim to identify those refactorings which could be applied in order to improve the system’s design accordingly with the studied design flaw. Taking into account that classes with high values of metrics CC and Ce are hard to understand and maintain, we then identify a set of possible refactoring strategies to be applied on the list of suspect entities. In this paper, our study refers to ”Extract Method” refactoring method. After the refactoring is applied, we obtaind a new design of the system, design that will be again evaluated. The metrics values obtained before and after applying this refactoring are comparatively analyzed.

Table 1. Project 01 - The list of classes' clusters with the corresponding metrics

Cluster	Class Id	CC	EC	LCOM
1.1	1, 3, 4, 5, 7, 10, 11,15, 16, 17, 19, 21, 23, 24, 25, 26, 27, 28, 30, 31	medium	medium	high
1.2	2, 6, 9, 12, 20, 29	small	small	small
2	8, 13, 14, 32, 18, 22	high	high	high

3. Case Study

A case study was used to validate our metric based approach for identifying God Class design flaw. Two medium size projects, having 32 classes respectively 52 classes, were analyzed. The selected metrics are those described in Section 2.2, following the heuristics that characterize the "God Class" design flaw, i.e. Cyclomatic Complexity (CC), Lack of Cohesion Of Methods (LCOM) and Efferent Coupling at type level (Ce). The metrics were computed using NDepend [10] software.

Therefore, the first two steps (setting the assessment objectives and metrics identification) that define the proposed methodology for OOD assessment (Section 2) have already been performed. We have also mention that the assessment objectives and the selected metrics are the same for both projects.

In what follows we apply Fuzzy Divisive Hierarchic Clustering (FDHC) algorithm [3], computing also the DFE metric value, and we suggest some refactorings to improve the design. After the refactorings were applied, the newly resulted design was also assessed and some comparasions were made.

3.1. Project 01 - Case Study.

After applying the FDHC algorithm [3] we have obtained the clusters briefly described in Table 1.

The analysis of the metric's values from each cluster reveals the cluster with "problems" regarding the "God Class" design flaw: in the first cluster, i.e. 1.1 the CC metric and Ce metric have "good" values, but the LCOM value indicate a poorly cohesive class, for the 1.2 cluster all the metric values are "good", indicating that the classes from this cluster are not candidates for the "God Class' design flaw, but the elements from the cluster 2 have the value for the Cyclomatic Complexity metric greater than 30 (being extremely complex), Ce metric is high and the LCOM very closed to 1, indicating poorly cohesive class and have more than one responsibility. Thus, the classes from the 2 cluster are candidates for "God Class" design flaw, being candidates for refactoring.

TABLE 2. Project 01 - Metrics values before and after applying refactoring for the analyzed classes.

	Before			After		
Class	CC	Ce	LCOM	CC	Ce	LCOM
DataServer	37	28	0.86	35	28	0.64
IM	59	43	0.96	44	43	0.91
InvForm	52	67	0.94	51	67	0.94

TABLE 3. Project 02 - The list of classes' clusters with the corresponding metrics

Cluster	Class Id	CC	Ce	LCOM
1.1	3, 24, 25, 26, 29, 44, 45, 47	medium	medium	0
1.2	1, 2, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 22, 23, 27, 28, 30, 31, 32, 33, 34, 36, 37, 38, 39, 41, 42, 43, 46, 48, 49, 50, 51, 52	small	small	0
2.1	20,21,35,40	high	high	high
2.2	10, 18, 19	medium	medium	high/0

The value of the DFE metric is 1.33, the minimum value being 0.4 and the maximum value 1.58, the obtained value suggesting that the "God class" design flaw is largely studded into the software system.

The suggested refactoring are Extract Method and this should be applied for the classes in the 2 cluster.

The Extract Method was applied for the following 3 classes: DataServer, IM and InvForm, respectively 7 methods, and the complexity of some methods/classes was reduced. See Table 2for more details.

3.2. **Project 02 - Case Study.** Regarding the second project we analyzed, after applying the FDHC Algorithm [3] we have obtained the clusters specified in Table 3.

The analysis of the metric's values from each cluster reveals the cluster with "problems" regarding the "God Class" design flaw: in the first (1.1.) and the second (1.2) cluster the values of the metrics don't indicate "God Class" design flaw entities, the third cluster 2.2 has "bad" values for each metric (high values for CC and Ce, and value near 1 for the LCOM metric), cluster 2.2 hhas "goog' value for the *Ce* metric but "bad" values for the other two

TABLE 4. Project 02 - Metrics values before and after applying refactoring for the analyzed classes.

	Before			After		
Class	Ce	CC	LCOM	Ce	CC	LCOM
LinkMemoDbDataSet	35	59	0.82	37	59	0.83
TableAdapterManager	63	43	0.78	66	43	0.81
tbl_UrlTableAdapter	55	48	0.9	43	48	0.92

metrics, and the cluster with the isolation points, i.e. IP contains classes with "good" values for all the metrics, except for the CC metric (but not very high).

The value of the DFE metric is 1.07, having minimum value 0.33 and the maximum value 1.37, the obtained value suggesting that the "God class" design flaw is largely studded into the software system.

Extract Method refactoring was applied for three classes from the 2.1 cluster: LinkMemoDbDataSet, tbl_UrlTableAdapter, TableAdapterManager. See Table 4for more details.

Analyzing the results obtained after applying the proposed refactoring for the second project, we can conclude that:

- there is only one improvement, the class *tbl_UrlTableAdapter* decreases its complexity
- the value of Ce metric remains unchanged for the refactored classes
- the values of metrics CC and LCOM metrics have higher values (except for *tbl_UrlTableAdapter* class) which does not improve the design.

So, if for the first project we obtained an improvement after applying refactoring, for the second one we have to consider a new analysis to suggest other refactoring.

The decision of choosing a refactoring and the analysis of the impact of applying it, is compromise decision, known in literature as "technical debt" cite Suryanarayana14. Technical debt is a recent metaphor referring to the eventual consequences of any system design, software architecture or software development within a codebase. It is possible that choosing a particular refactoring, to improve some quality attributes, while others remain the same or even worse.

4. Conclusion

Software metrics are considered of great importance in software quality assurance but there is a gap between the things measured and the ones really important in terms of quality characteristics. The current paper proposes a new metric based approach to evaluate the impact of applied refactoring on software desing.

For future work, we intend to focus our research in the following directions:

- to apply the proposed evaluation framework on more case studies;
- to automate the task of establishing the list of suspect entities and the list of refactorings that could be applied.

References

[1] R. Marinescu: *Measurement and quality in object-oriented design*, Ph.D. thesis in the Faculty of Automatics and Computer Science of the Politehnica University of Timisoara, 2003.

[2] Camelia Serban, *A Conceptual Framework for Object-oriented Design Assessment*, Computer Modeling and Simulation, UKSim Fourth European Modelling Symposium on Computer Modelling and Simulation, 90–95, 2010.

[3] Dumitrescu, D., *Hierarchical pattern classification*, Fuzzy Sets and Systems 28, 145–162, 1988.

[4] A.J. Riel. *Object-Oriented Design Heuristics*, Addison-Wesley, 1996.

[5] M. Salehie and Li Shimin and L. Tahvildari *A Metric-Based Heuristic Framework to Detect Object-Oriented Design Flaws*, Program Comprehension, 2006.

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

[7] C. Serban and A. Vescan and H.F. Pop *Design Flaw Entropy metric for software quality assessment*, Computer Science - Research and Development (submitted)

[8] S.R. Chidamber and C.F. Kemerer. *A Metric Suite for Object- Oriented Design.* IEEE Transactions on Software Engineering, 20(6):476493, 1994.

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

[10] NDepend, *NDepend, http://www.ndepend.com/*, 2015.

[11] T.J. McCabe. *A Complexity Measure.* IEEE Transactions on Software Engineering, 2(4), pages 308320, 1976.

[12] Arthur H. Watson and Thomas J. McCabe. *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric.* In National Institute of Standards and Technology NIST Special Publication, pages 500235, 1996.

[13] Suryanarayana G. *Refactoring for Software Design Smells (1st ed.).* Morgan Kaufmann. p. 258. ISBN 978-0128013977. 2014

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: motogna@cs.ubbcluj.ro

E-mail address: camelia@cs.ubbcluj.ro

E-mail address: avescan@cs.ubbcluj.ro