Extracting and Modeling Design Defects Using Gradual Rules and UML Profile

Mohamed Maddeh, Sarra Ayouni

Extracting and modeling design defects using gradual rules and UML profile

Mohamed Maddeh[1] and Sarra Ayouni[2]

[1]SOIE, ISG Tunis, Le Bardo, Tunis –Tunisia
maddeh_mohamed@yahoo.com
[2]Faculty of Sciences of Tunis, Tunisia
s_ayouni@yahoo.fr

Abstract. There is no general consensus on how to decide if a particular design violates a model quality. In fact, we find in literature some defects described textually, detecting these design defects is usually a difficult problem. Deciding which object suffer from one defect depends heavily on the interpretation of each analyst. Experts often need to minimize design defects in software systems to improve the design quality. In this paper we propose a design defect detection approach based on object oriented metrics. We generate, using gradual rules, detection rules for each design defect at model level. We aim to extract, for each design defects, the correlation of co-variation of object oriented metrics. They are then modeled in a standard way, using the proposed UML profile for design defect modeling. We experiment our approach on 16 design defects using 32 object oriented metrics.

Keywords: Object oriented metrics, Data Mining, Gradual rules, Design defects detection, UML profile.

1 Introduction

Design defects which are also called design anomalies, refer to design situations that adversely affect the development of software like bad smells [9] and antipatterns [2]. The first one (i.e., bad smells) was proposed by Beck [9]. In fact, the author defines 22 sets of symptoms of common defects. The second one (i.e., anti-patterns) was introduced by Brown et al. [2]. A set of refactoring suggestions are associate for each defect type. Detecting these defects at the model level is a promising way to improve software maintenance process [4][6][21]. In addition, it is difficult to identify and express these anomalies as rules [17], since they are not formalized and based on a simple textual description.

In general, design defects are evaluated using rules in the form of metric/threshold combinations. Some works propose rules manually identified [1][17], other propose algorithms that generate these rules[5][11][14]. Both approaches are suffering from two major difficulties. The first one is due to the large number of possible metrics combinations, in fact, it is difficult to find the best suitable rule. The second problem

is to find the best threshold for each metric. In this paper, we propose a predictive design defects detection that focuses on model level in order to correct them before there propagation to the code. Also, instead of affecting a threshold for metrics, we generate, using gradual rules a correlation of co-variation of metrics characterizing the object oriented design defects. We model each defect using UML profile, defect are then represented as an UML class diagram summarizing the relevant information from the most significant textual descriptions in literature.

The remainder of the paper is structured as follows. In section 2, we present the related works. In section 3, we give the problem statement. In Section 4, we introduce the general process of the approach. In sections 5, we validate the proposed approach and section 6 is reserved for conclusion.

2 Related works

Several studies have recently focused on detecting design defects in software using different techniques. In [14] authors propose a new framework M-RAFACTOR for the detection and correction of design defects based on object oriented metrics. Marinescu [9] defined a list of rules relying on metrics to detect what he calls design flaws of OO design at method, class and subsystem levels. Erni et al. [18] use metrics to evaluate frameworks with the goal of improving them. Another model refactoring is presented by Marc Van Kempen et al. [13], based on SAAT (Software Architecture Analysis Tool). It allows calculating metrics about UML models the metrics are then used to identify the flaws or anti-patterns. Authors represent the structure using class diagrams, and the behaviour of each class using statecharts. After that they examine the metrics for refactoring a centralized control structure into one that employs more delegation. For the four previous contributions it is difficult to manually define threshold values for metrics in the rules.Moha et al. [15], in their DÉCOR approach, they start by describing defect symptoms using an abstract rule language. These descriptions involve different notions, such as class roles and structures. In [11] defect detection is considered as an optimization problem. They propose an approach for the automatic detection of potential design defects in code. The detection is based on the notion that the more code deviates from good practices, the more likely it is bad.

3 Problem Statement

There are many open issues that need to be addressed when detecting design defects. In this paper, we first focus on how to define detection rules when dealing with quantitative information and then how to give a unified representation of defects specifications.

In fact, we notice that the textual description of design defects presented by authors depend on a subjective interpretations of analysts. As fact, for a same design we can find variable set of defects depending on the criteria's used by designer team. To bridge the gap between the description and the detection process, each design defect must be formalized for the standardization of the definition of symptoms detection. In

this paper we intend to use gradual rules to formalize design defects. In the context of our research the generated gradual rules are represented as a correlation of co-variation of object oriented metrics. Once, gradual rules identified each design defect is then modeled using the UML profile for design defects. We have proposed an UML profile for design defect modeling. It summarizes the most relevant information and replaces all textual descriptions existing in literature by one class diagram for each design defect.

4 The General process

As presented in figure 1, we start with the domain analysis of the knowledge extracted from the textual description of design defects. In fact, domain analysis is a process in which information used in developing software systems is identified, captured, and organized to be reusable when creating new systems [8]. In our context, information about design defects must be well structured and reusable for the automated detection process. Thus, we have studied the textual descriptions of design defects. We present an antipattern example named the Blob.

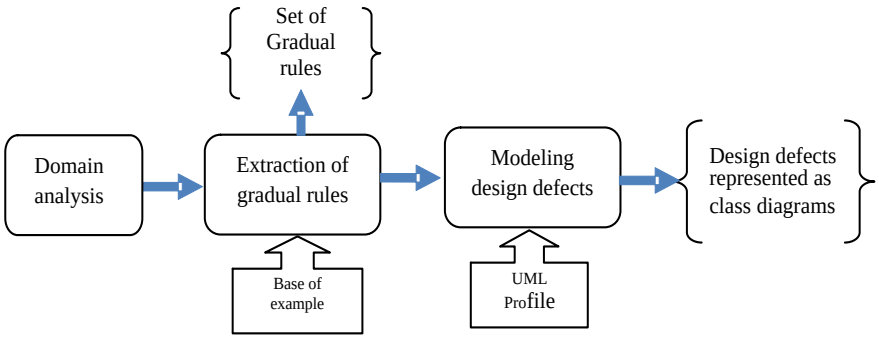

Fig. 1. General process

The Blob (called also God class [16]) corresponds to a large controller class that depends on data stored in surrounded data classes. A large class declares many fields and methods with a low cohesion. After the domain analysis for the Blob antipattern, we extract the relevant information. Indeed the blob is an interclass and behavioral defect, related to static and behavioral diagrams. The detection of the blob is based on the analysis of the class diagram and the sequence diagrams. As presented in table1, this research is based on 16 design defects.

	Blob	SwissArmyKnife	Lava Flow	Poltergeists	FunctionalDecomposition	God Package	God Classes	Long Parameter List	Data Clumps	Divergent Change	ShotgunSurgery	Lazy Classes	FeatureEnvy	Comments	Data Classes	RefusedBequest
Structural					*			*	*							
Semantic														*		
Behavioral	*	*	*	*		*	*			*	*	*	*		*	*

Table 1. Classification of design defects

These design defects are evaluated using object oriented metrics that are also identified at this step. Metrics must be measurable at model level, and useful for detection process. In our work we have identified 32 metrics. In what follows, we present some of these metrics:

Access To Foreign Data (ATFD) [12] represents the number of external classes from which a given class accesses attributes, directly or via accessor-methods.

Weighted Method Count (WMC) [3] is the sum of the complexity of all methods in a class.

Attribute Per method (APM) is defined as the ratio of the metrics Number of attributes (NOA) and (NOM).

After the metrics identification step we extract for each defect the most significant gradual rules that express the correlation of co-variation of the object oriented metrics. We propose an approach that uses knowledge from previously manually inspected projects, called defects examples.

4.1 Mining gradual rules

In our research, gradual rules are used to evaluate poor design by detecting bad smells and antipatterns. Mining gradual rule has been extensively used in fuzzy command systems. However, in last decade, the data mining community has been interested in extracting such kind of rules [7] [10] [19] [20]. Gradual rule convey knowledge of the form « the more/the less A, the more/the less B ». In our context, A and B are object oriented metric. We thus propose to extract rules such as « the more/the less Metrique1, the more/the less Metrique2…, the more/the less Metrique n », such that these metrics characterize a defect X. To the best of our knowledge, no previous study in the literature has paid attention to apply the extraction of gradual rules to the design defects detection. In the following section, we recall the key concepts of gradual rules mining.

Gradual Rules

We consider a data base defined on a schema containing m attributes (X1, …,Xm) defined on domains dom(Xi) provided with a total order. A data set D is a set of m-tuples of dom(X1),…,dom(Xm). In this scope, a gradual item is defined as a pair of an attribute and a variation {+,-}.The gradual item Xn+, means that the attribute Xn is increasing. It can be interpreted by the more A. A gradual itemset, or gradual tendency, is then defined as a non-empty set list of several gradual items.

For instance, the gradual itemset M =A+ B– is interpreted as, the more A and the less B. For example, the relation from Table 2 shows various items about disease symptoms.

	Patients	Temperature	Lymphocyte	Hemoglobin
T1	P1	37.8	32	14
T2	P2	38.2	17	10
T3	P3	38.1	15	16

Table 2. Disease symptoms

This table contains three tuples : {T1,T2,T3}, we study co-variations from one item to another one, as for example the variation of the temperature and hemoglobin. Too kinds of variations are considered: increasing variation and decreasing variation. Each item will hereafter be considered twice: once to evaluate its increasing strength, and once to evaluate its decreasing strength, using the + and - operators.

For example, let us consider the rule "The higher temperature and the higher hemoglobin then lower the lymphocyte" formalized by : R1= (Temperature + Hemoglobin + Lymphocyte -).

4.2 Mining gradual design defect rules

In this section, we present the extraction of gradual design defects rules. It is based on the GRITE algorithm [10], for GRadualITemset Extraction. For each design flaw, we identify the metric-based heuristics. The majority of works assign a threshold to each metric. The quality of the solution depends on the number of detected defects in comparison to the expected ones in the base of examples. The main limitation of this approach is that it is difficult to find the best threshold.

To overcome this problem, we present another type of correlation between object oriented metrics. To do so, we associate for each defect a metrics table; it represents the different metrics values for each occurrence (Oi) of all defects extracted manually from various projects (Pi). As example, we present in table 3 a part of the metrics table for the defect Data Class. The Data class defect creates classes that passively store data. Classes should contain data and methods to operate on that data.
Where, for a given class C we have:

PS: Package Size, NC: Number of Classes in the model, NOPM: Number Of Packages in the Model, NOC: Number Of Communications, is the number of messages sent by the class C, NMSC: Number Of Messages for the Same Class, is the number

of internal messages from C to C, NCC: Number of Connected Classes, is the number of classes that communicates with the class C and NCM: Number of connected messages, is the number of messages sent to the class C.

		ATFD	NOM	NOA	PS	NC	NOPM	NOC	NMSC	NCC	NCM
P1	O1	03	15	08	22	57	02	05	03	02	01
	O2	02	10	05	28	57	02	04	02	01	00
P2	O3	04	08	10	33	113	04	06	09	00	01
	O4	02	13	07	33	113	04	04	07	04	02
	O5	05	14	08	25	113	04	07	08	04	06
	O6	06	09	11	24	113	04	08	04	06	05
	O7	04	16	13	21	113	04	04	07	09	08
P3	O8	05	17	12	52	368	11	06	06	03	04
	O9	02	13	12	46	368	11	04	05	05	02

Table 3. Data Class metrics

The GRITE algorithm gives the most frequent sequences of metrics using the minsupport threshold. Where, the minsupport threshold aims at discovering subsets of items that occurs together at least a minsupport time in a database. If minsupport is set to be too large, no itemsets will be generated, if minsupport is set to be too small, huge number of itemsets will be generated. Fixing the minsupport threshold depend on the specificities of the problem.

In the context on design defect detection, almost we don't have a very large database comparing to other domains, that's why we set a minsupport value to be more than 0.5. It means that we will extract the gradual rules that occur at least in 50% of the transactions. We can decrease the minsupport thresholds if the program generates no rule, until having at least one rule.

4.3 Modeling design defects

Based on UML profile capabilities, we extend the UML metamodel to support and model all key concepts used for the specifications of design defects. We model each defect to create a catalogue of design flaws. We formalize a set of textual and informal design flaws description (avoiding any subjective interpretation) in a well-structured model enclosing all necessary information to deal with design defect detection.

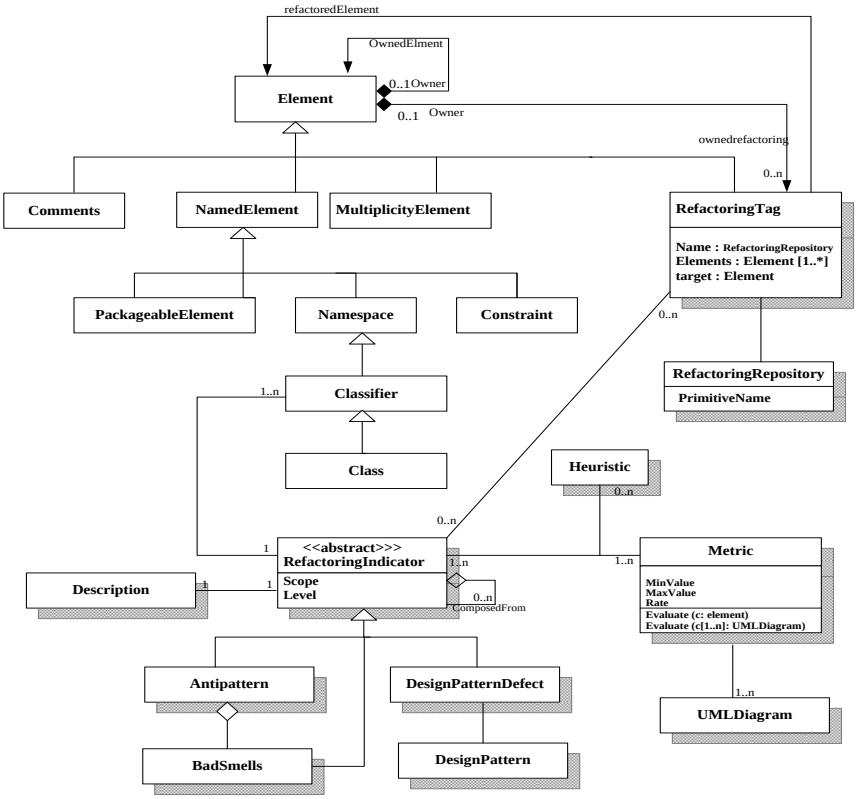

Fig. 2. The UML profile

Defined Stereotypes

In this section, we detail the defined stereotypes illustrated in figure 2: RefactoringIndicator is a super-class modeling all possible refactoring indicators. The design flows can be specialized as Antipattern, DesignPatternDefect, BadSmells.

Description contains a textual description of the design flaws. It represents the semantic aspect. The description stereotype is very helpful to understand the meaning of the design defect and the context in which it can be identified.

Metric represents the set of metrics useful forsoftware measurement and design flows detection. The measure of metrics is done over the static and/or dynamic UML diagrams. The UMLDiagram stereotype represents the UML diagrams attached to the metric concept. Each Design pattern defect is attached to a design pattern represented by the stereotype DesignPattern. RefactoringRepository indicates the name of the refactoring primitive, using the attribute PrimitiveName (For the design defects correction).

Figure 3 in the next section presents the UML class diagram of the Data Class defect.

5 Experiments

The experimentations concern the defects Blob, Lazy class, Data class, Feature Envy and Lava flow, using three minsupport thresholds 0.5, 0.8 and 0.9. The Lazy class defect occurs when class isn't doing enough to pay for itself. Every additional class increases the complexity of a project. The Feature Envy defect occurs for methods that make extensive use of another class and may belong in another class. And the Lava flow defect represents the model elements that are not really used in the project due to an overestimation of needs.

		Minsupport	
		0.5	**0.8**
Blob	***R1***	(ATFD+ PS+ NC+ NOPM-)	No Rule
	R2	(ATFD+ NOM+ NOA- NCM+)	
	R3	(ATFD+ NC+ NCM+ NCC+ NOM+)	
	R4	(ATFD+ NMSC+ NOA- PS+ NC+)	
Lazy class	***R1***	(NC+ NCC- ATFD- NCM- PS+)	(NCM- NOM – NC+ NCC- ATFD-) (NC+ NCC- ATFD- NCM- PS+)
	R2	(NCM- NOM – NC+ NCC- ATFD-)	
	R3	(ATFD- NC+ NCM- NOPM- NOM-)	
Data class	***R1***	(APM- ATFD- NC+ PS+ NCC-)	(NOM- PS+ NCC- NC+ NCM+)
	R2	(NOM- PS+ NCC- NC+ NCM+)	
FeatureEnvy	***R1***	(NIC+ NMSMC- NC+ PS+ NOPM-)	No Rule
Lava flow	***R1***	(NC+ NCC- ATFD- NCM- PS+)	(NCM- NOM – NC+ NCC- ATFD-)
	R2	(NCM- NOM – NC+ NCC- ATFD-)	(NC+ NCC- ATFD- NCM- PS+)
	R3	(ATFD- NC+ NCM- NOPM- NOM-)	(NIC- CM- APM- NC+ NOPM+)
	R4	(NIC- NMSMC- CM- NOM+)	
	R5	(NIC- CM- APM- NC+ NOPM+)	

Table 4. Results

The Table 4 presents our results. We choose to select gradual design defect rules that contain more than four metrics to guaranty significant results avoiding an overwhelming rule set. These results indicate the common conditions for the occurrence of a design defect. We have better rules for high minsupport value 0.8, because the rule is repeated at the majority of the defect occurrence (80%). We have lowest minsupport threshold 0.5 guaranty that the extracted rules occurs in at least 50% of the detected defects.

In our case, for a minsupport threshold equal to 0.9 we have no rules for all defects. We notice that the activity of design defects detection depends on the subjectivity of the designer. In fact, our research intends to help designer to improve the quality of models by offering a set of gradual rules characterizing the context in which could occur a design defect. All important information related to defects is now represented using the UML profile. In figure 3 we present an example for the Data Class defect.

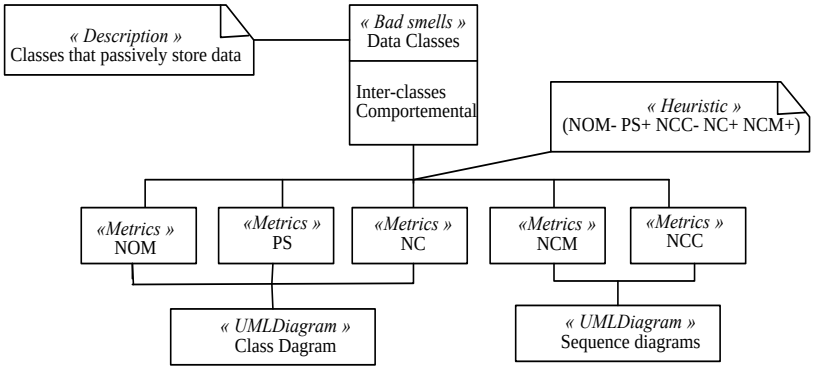

Fig. 3. Data Classes

6 Conclusion

Several design defect detection techniques have been proposed. Most of existing works relies on metrics rule-based detection, applied for the code level. However, it is difficult to identify and express these symptoms as rules [17], since they are not formalized. It is also difficult to find the best threshold for metrics. This work raised some interesting perspectives in order to detect design defects for model level based on the evaluation of correlation of metrics co-variation instead of threshold. We have also proposed an UML profile for design defect modeling. It fully supports design defects modeling needs. It allows antipatterns and bad smells modeling with one unified language. Using the UML profile for design defects, we unify software designer teams with a single and shared design defects specification.

References

1. Abreu, F. B. E. et Melo, W. (1996). Evaluating the impact of object-oriented design on software quality. *In The 3 rd International SoftwareMetrics Symposium*, pages 90–99.
2. Brown, W. J., Malveau, R. C., McCormick, H. W. S. et Mowbray, T. J. (1998b). AntiPatterns : Refactoring Software, Architectures, and Projects in Crisis : Refactoring Software, Architecture and Projects in Crisis. John Wiley & Sons.
3. Chidamber, S., Kemerer, C.: A metrics suite for object oriented design. IEEE Transactions on Software Engineering 20(6) (1994) 476–493.
4. Corradini, A., H. Ehrig, H.-J. Kreowski and G. Rozenberg, editors,"Graph Transformation" Lecture Notes in Computer Science 2505, Springer-Verlag, 2002.
5. Erni, K. et Lewerentz, C. (1996). Applying design metrics to object-oriented frameworks. *In IEEE METRICS*, pages 64–74.
6. Ethan Hadar, Irit Hadar, "The Composition Refactoring Triangle (CRT) Practical Toolkit: From Spaghetti to Lasagna", OOPSLA 2006, Portland, Oregon, USA. ACM 1-9593-491-X/06/0010.
7. E. Hüllermeier. Association rules for expressing gradual dependencies. In Proceedings of the 6th European Conf. on Principles of Data Mining and Knowledge Discovery, PKDD'02, pages 200–211. Springer-Verlag, 2002.

8. Frakes, W., Prieto-Diaz, R., & Fox, C. (1998). “DARE: Domain Analysis and Reuse Environment”. Annals of Software Engineering (5), , pp. 125-141.
9. Fowler, M., Beck, K., Brant, J., Opdyke, W. et Roberts, D. (1999). Refactoring : Improving the Design of Existing Code.
10. L. Di Jorio, A. Laurent, and M. Teisseire. Mining frequent gradual itemsets from large databases. In Proceedings of the Int. Conf. on Intelligent Data Analysis, IDA’09, Lyon France,2009.
11. Marouane Kessentini, Wael Kessentini, Houari Sahraoui, Mounir Boukadoum and Ali Ouni. Design Defects Detection and Correction by Example, 19th IEEE International Conference on Program Comprehension, 2011.
12. Marinescu. Detecting Design Flaws via Metrics in Object-Oriented Systems. In Proceedings of TOOLS USA 2001, pages 103–116. IEEE Computer Society, 2001.
13. Marc Van Kempen, Michel Chaudron, Derrick Kourie, Andrew Boake, “Towards Proving Preservation of Behaviour of Refactoring of UML Models”, in proceedings of SAICSIT 2005, Pages 252.
14. Maddeh Mohamed, Mohamed Romdhani, Khaled Ghedira, M-REFACTOR: A New Approach and Tool for Model Refactoring, ARPN Journal of Systems and Software, JULY 2011.
15. N. Moha, Y.-G. Guéhéneuc, L. Duchien, and A.-F. L. Meur: DECOR: A method for the specification and detection of code and design smells, Transactions on Software Engineering (TSE), 2009, 16 pages.
16. Riel, A.J.: Object-Oriented Design Heuristics. Addison-Wesley (1996).
17. Radu Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In Proceedings of the 20th International Conference on Software Maintenance, pages 350–359. IEEE Computer Society Press, 2004.
18. Raul Marticorena and Yania Crespo. Refactorizaciones de especializacion sobre el lenguaje modelo MOON. Technical Report DI-2003-02, Departamento de Informatica. Universidad de Valladolid, septiembre 2003.
19. Sarra Ayouni, Sadok Ben Yahia, Fuzzy set-based formalization of gradual terns, SoCPaR, 2014: 434-439, Tunis, Tunisia.
20. Sarra Ayouni, A. Laurent, S. Ben Yahia and P. Poncelet. Fuzzy gradual patterns: What fuzzy modality for what result? In Proceedings of the International Conference on Soft Computing and Pattern Recognition (SoCPaR’10), Cergy, France 2010.
21. Zhang, J., Lin, Y. and Gray, J. (2004) “Generic and Domain-Specific Model Refactoring using a Model Transformation Engine”, Model-driven Software Development – Research and Practice in Software, 2005.