

Detection and Correction of Design Defects in Object-Oriented Architectures

Naouel Moha

GEODES – Research Group on Open, Distributed
Systems, Experimental Software Engineering
Department of Informatics and Operations Research
University of Montreal, Quebec, Canada
`mohanaou@iro.umontreal.ca`

Abstract. Design defects are poor design choices that lessen the quality of object-oriented architectures and impede their evolution and their maintenance. A good architecture without defects reduces significantly maintenance costs by easing comprehension and changes. However, the detection and correction of design defects are difficult because of the lack of precise specifications of defects and tools for their detection and correction. Our goal is to provide a systematic method to specify design defects precisely and to generate detection and correction algorithms using refactorings from their specifications semi-automatically. We propose to apply and validate these algorithms on open-source object-oriented programs to show that our method allows the systematic description, detection, and correction of design defects with a reasonable precision.

1 Introduction

Large object-oriented programs are expensive to maintain because of bad design practices and architectural drift [1], which make adding, debugging, and evolving features difficult. These bad design practices are the root causes of *design defects*. Design defects are bad solutions to recurring design problems in object-oriented architectures, which introduce problems in the design and have negative consequences on maintenance. They encompass problems with different granularities, from architectural (global) problems, such as antipatterns [2], to low-level (local) problems, such as code smells [4] (such as long methods, long parameter lists, or large classes).

The Blob (called also God class [5]) is a typical example of a design defect. It corresponds to a large controller class that depends on data stored in interrelated data classes. A large class declares many fields and methods with a low cohesion. A controller class monopolises most of the computation done by a program, takes most of the decisions, and closely directs the processing of other classes. A data class contains only data and performs no processing on these data. It is composed of highly cohesive fields and accessors.

A good software architecture without design defects is easier to understand and change and thus maintain. However, the detection and correction of defects

in large architectures remain manual activities for lack of tools because defects affect different classes and methods and are based essentially on textual descriptions subject to different interpretations. When more formal definitions are given, they are often only UML-like diagrams with few semantics and only represent one possible variation of the defects. Consequently, the detection and correction of design defects in large architectures are highly time- and resource-consuming and error-prone activities.

In the face of these difficulties, researchers have proposed processes and techniques to detect design defects [7,8,9,10] and correct them with refactorings [2,4,5]. Yet, existing processes and techniques have several limitations. Processes are mostly based on manual inspections and therefore do not scale to large programs easily and detection techniques mainly use software metrics, which cannot reflect the whole complexity of the design of an architecture. In particular, the imprecision of the textual descriptions hinder the understanding of the design defects and the implementation of detection and correction algorithms.

2 Approach

Our goal is to provide a systematic method to specify design defects precisely and to generate detection and correction algorithms from their specifications semi-automatically. We cover the entire process from the detection to the correction of design defects because, until now, these two related activities have been studied separately, and we enrich the detection of defects with not only metric-based heuristics but also semantic and structural information. This method consists in designing a meta-model to model design defects using a taxonomy of the defects and in generating detection and correction algorithms from the models [11]. Our method decomposes in 10 steps (*cf.* Figure 1).

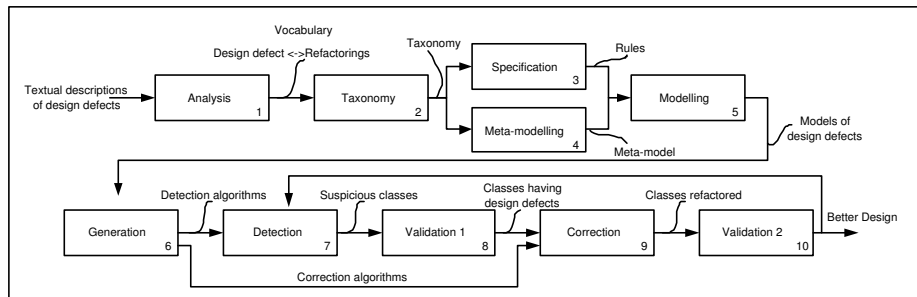


Fig. 1. A Systematic Method for the Detection and the Correction of Design Defects.

1. Analysis: We extract key concepts from the textual descriptions of design defects in the literature. Key concepts include metric-based heuristics as well as

structural and semantic information to form a unified vocabulary of reusable concepts to describe and correct design defects. At this step, we associate manually with each design defect the heuristics to detect them and the set of refactorings to correct them.

2. *Taxonomy:* We define a taxonomy of the described design defects by classifying their key concepts in disjoint (sub)categories [11]. This taxonomy is a reference model to distinguish design defects, to highlight their commonalities and specificities, and thus, to avoid misunderstanding and misinterpretations.

3. *Specification:* We specify a language using a set of rules and compositions thereof that allows to describe design defects synthetically and their associated refactorings based on the unified vocabulary defined in the first step.

4. *Meta-modelling:* We design a meta-model to instantiate rules that specify design defects. This meta-model offers as constituents all the key concepts used in the specifications of design defects. We use this meta-model to build models of the rules that can be manipulated programmatically, in particular to generate detection and correction algorithms automatically. This meta-model also provides a parser to analyse and to reify rules concretely.

5. *Modelling:* We instantiate the meta-model to build concrete models of design defects that can be manipulated programmatically. The set of modelled design defects forms a catalogue, which is the basis for the detection and correction techniques.

6. *Generation:* We generate detection and correction algorithms from the modelled design defects using the constituents of the meta-model (and associated rule cards and key concepts). The algorithms are based on metric values and on the semantic and structures of architectures.

7. *Detection:* We apply the generated detection algorithms on open-source programs to detect suspicious classes that have potential design defects.

8. *Validation 1:* We validate the generated algorithms with the community and their results manually ourselves and with the developers by assessing the precision and the recall.

9. *Correction:* We apply the generated correction algorithms on the set of classes validated as having design defects.

10. *Validation 2:* The maintainers apply or not the refactorings proposed by the correction algorithms to obtain a better design. The correction of defects is validated concretely by ensuring that defects corrected have disappeared and that their correction has not introduced other defects, thus the method is iterative.

3 Conclusion

The (semi-)automated detection and correction of design defects is an important issue for improving the quality of programs, facilitating their evolution and their maintenance and thus reducing their overall cost. However, existing textual descriptions of design defects are informal, subject to (mis)interpretations, and the manual detection and correction of design defects is a tedious and time-consuming activity. We introduced a systematic method to describe design defects precisely and to generate automated detection and correction algorithms. This method allows maintainers to specify design defects easily in terms of structural, semantic, and measurable properties of classes and structural relationships among classes. It allows the systematic specification and detection of design defects, in contrast with previous works, which provide ad hoc detection and correction algorithms only. Moreover, this method facilitates the development of concrete tools for the detection and correction of design defects.

Acknowledgements. I am deeply grateful to my supervisor, Yann-Gaël Guéhéneuc, for his guidance, support, and encouragement.

References

1. Perry, D.E., Wolf, A.L.: Foundations for the study of software architecture. *Software Engineering Notes* **17** (1992) 40–52
2. Brown, W.J., Malveau, R.C., Brown, W.H., III, H.W.M., Mowbray, T.J.: *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*. 1st edn. John Wiley and Sons (1998)
3. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns – Elements of Reusable Object-Oriented Software*. 1st edn. Addison-Wesley (1994)
4. Fowler, M.: *Refactoring – Improving the Design of Existing Code*. 1st edn. Addison-Wesley (1999)
5. Riel, A.J.: *Object-Oriented Design Heuristics*. Addison-Wesley (1996)
6. Hanna, M.: Maintenance burden begging for a remedy. *Datamation* (1993) 53–63
7. Marinescu, R.: Detection strategies: Metrics-based rules for detecting design flaws. In: *Proceedings of the 20th International Conference on Software Maintenance*, IEEE Computer Society Press (2004) 350–359
8. Munro, M.J.: Product metrics for automatic identification of “bad smell” design problems in java source-code. In Lanubile, F., Seaman, C., eds.: *proceedings of the 11th International Software Metrics Symposium*, IEEE Computer Society Press (2005)
9. Travassos, G., Shull, F., Fredericks, M., Basili, V.R.: Detecting defects in object-oriented designs: using reading techniques to increase software quality. In: *proceedings of the 14th conference on Object-Oriented Programming, Systems, Languages, and Applications*, ACM Press (1999) 47–56
10. Trifu, A., Dragos, I.: Strategy-based elimination of design flaws in object-oriented systems. In: *proceedings of the 4th international Workshop on Object-Oriented Reengineering*, Universiteit Antwerpen (2003)
11. Moha, N., Huynh, D.L., Guéhéneuc, Y.G.: Une taxonomie et un métamodèle pour la détection des défauts de conception. In Rousseau, R., ed.: *actes du 12^e colloque Langages et Modèles à Objets*, Hermès Science Publications (2006) 201–216