

Detection and Correction of Design Defects in Object-Oriented Designs

Naouel Moha

Ptidej Team – GEODES Group, DIRO
University of Montreal, QC, Canada
mohanaou@iro.umontreal.ca

Abstract

Design defects come from poor design choices and have the effect of degrading the quality of object-oriented designs. Therefore, they present opportunities for improvements. However, design defects have not been precisely specified and there are few appropriate tools that allow their detection as well as their correction. Our goal is to provide a systematic method to specify systematically design defects precisely and to generate automatically detection and correction algorithms from their specifications. The detection algorithms are based not only on metrics but also on semantical and structural properties whereas the correction algorithms are based on refactorings. We apply and validate these algorithms on open-source object-oriented programs to show that our method allows a systematic specification, a precise detection, and a suitable correction of design defects.

Category and Subject Descriptors D.2.7 Software engineering; Distribution, maintenance, and enhancement; Restructuring, reverse engineering, and reengineering

General Terms Algorithms, design, languages.

Keywords Design defects, antipatterns, code smells, specification, meta-modelling, detection, correction, refactorings.

1. Research Context: *Software Quality*

High-quality software is an important goal in the software development process because software are everywhere from the game applications to embedded systems such as navigation systems in aerospace. Software quality is assessed and improved mainly during formal technical reviews, which primary objective is to detect errors and defects early in the software process, before they are passed on to another software engineering activity or released to the customer [7]. High-quality software is harder to achieve in large object-oriented programs since they are expensive to maintain be-

cause bad design practices are the root causes of *design defects*, which make it difficult to add, debug, and evolve features.

We define design defects as wrong solutions to recurring problems in object-oriented designs, typically UML class diagrams. They encompass problems at different levels of granularity, ranging from architectural problems, such as antipatterns [1] (as opposed to design patterns [3]), to low-level problems, such as code smells [4] (symptoms of design defects). Code smells are possible symptoms of higher-level defects such as antipatterns.

A typical example of a design defect is the Spaghetti Code antipattern, which is characteristic of procedural thinking in object-oriented programming. Spaghetti Code is revealed by classes with no structure, declaring long methods with no parameters. The names of the classes and methods may suggest procedural programming. Spaghetti Code does not exploit (and prevents the use of) object-orientation mechanisms, such as polymorphism and inheritance.

2. Motivation: *Reduce Software Development Costs*

The detection and correction of design defects early in the process substantially reduce the cost of subsequent steps in the development and support phases [7]: Designs free of design defects are easier to implement, change, and maintain. However, their detection and correction in large designs are highly time- and resource-consuming and error-prone activities [10] because design defects crosscut classes and methods and their descriptions are subject to misinterpretation.

In the face of these difficulties, some researchers have proposed processes and techniques to detect design defects, yet they have several limitations. Processes such as [12] are mostly based on manual inspections and therefore do not scale easily to large programs. Some techniques mainly use text-based descriptions of design defects [1 ; 4 ; 8], which are unprecise. In particular, the imprecision of the text-based descriptions and the lack of precise and structured representation of design defects hinder the understanding of the design defects and the implementation of detection and correction algorithms. Current detection techniques [5 ; 6] mainly use software metrics and focus essentially on low-level defects such as code smells. However, metrics cannot reflect the whole complexity of the design of a program [13].

The solution advocated for correcting design defects is to apply refactorings [1 ; 4 ; 8]. Refactoring is a technique used to change “*the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior*” [4]. Whereas a lot of work has been done on refactorings, it is still unclear how to use them to correct defects and improve the code design [2]. Some approaches [9 ; 11] provide means to detect and to correct design defects but they are mainly based on metrics and do not apply to high-level design defects. However, no methodology or tool has been yet proposed for the detection and correction of high-level design defects.

The fundamental thesis of this work can be stated as follows: *How to automate the detection and correction of high-level design defects such as antipatterns?*

3. Our Global Approach: DECOR Method

To overcome the problems stated previously, we propose a method, called DECOR (*Defect dEtECTION for CORrection*), to specify systematically high-level design defects and to generate detection and correction algorithms from their specifications semi-automatically. Our method generalises previous work and it is built on our own experience. It subsumes previous work in a unified process and understanding. It consists of four main stages, from the analysis of the text-based descriptions of design defects through their reification to their detection and correction in programs:

1. **Specification:** We propose a taxonomy including the terminology and a classification of design defects to overcome the problem related to the text-based descriptions of design defects and avoid misinterpretation. We propose a specification of defects based on a meta-model to manipulate them programmatically and store them in a repository of high-level defects.
2. **Detection:** We define techniques and algorithms to detect design defects from the meta-model previously defined. These techniques and algorithms are based on some other properties related to the semantics and the structure of the design in addition to metrics. Thus, the precision in the detection of design defects will be enhanced. The algorithms and techniques developed have to be generic to detect any design defect.
3. **Correction:** We suggest refactorings to correct defects detected by identifying first semi-automatically the modifications to provide for improving the design. For the detection and correction, we propose to specify the techniques based on a rule-based language and a framework.
4. **Validation:** We set up experimental studies on various programs to validate our detection and correction results.

Thus, we refine text-based descriptions of design defects with precise specifications. We enhance previous metric-based approaches with semantical and structural properties,

which provides a greater expressive power. Finally, we propose a systematic method that provides means to specify and to generate detection and correction algorithms of high-level design defects.

Acknowledgements I am deeply grateful to my supervisors, Laurence Duchien, Yann-Gaël Guéhéneuc, and Anne-Françoise Le Meur, for their guidance and support.

References

- [1] W. J. Brown, R. C. Malveau, W. H. Brown, H. W. M. III, and T. J. Mowbray, *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*, 1998.
- [2] B. Du Bois, J. Verelst, and S. Demeyer, “Refactoring – Improving Coupling and Cohesion of Existing CodeDesign Patterns,” in *Proceedings of the 11th WCRE Conference*, 2004.
- [3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Software*, 1994.
- [4] M. Fowler, *Refactoring – Improving the Design of Existing Code*, 1999.
- [5] R. Marinescu, “Detection Strategies: Metrics-Based Rules for Detecting Design Flaws,” in *Proceedings of the 20th ICSM Conference*, 2004.
- [6] M. J. Munro, “Product metrics for automatic identification of “bad smell” design problems in java source-code,” in *Proceedings of the 11th Metrics symposium*, 2005.
- [7] R.S. Pressman, *Software Engineering – A Practitioner’s Approach*, 2001.
- [8] A. J. Riel, *Object-Oriented Design Heuristics*, 1996.
- [9] H. A. Sahraoui, R. Godin, and T. Miceli, “Can Metrics Help to Bridge the Gap Between the Improvement of OO Design Quality and Its Automation?,” in *Proceedings of the 16th ICSM Conference*, 2000.
- [10] B. Sapsomboon, “Shared Defect Detection : The Effects of Annotations in Asynchronous Software Inspection,” PhD thesis, University of Pittsburgh, 2000.
- [11] L. Tahvildari and K. Kontogiannis, “Improving design quality using meta-pattern transformations: A metric-based approach,” in *Journal of Software Maintenance*, (16)4-5, 2004.
- [12] G. Travassos, F. Shull, M. Fredericks, and V. R. Basili, “Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality,” in *Proceedings of the 14th OOPSLA Conference*, 1999.
- [13] S. M. Yacoub, H. H. Ammar, and T. Robinson, “Dynamic Metrics for Object Oriented Designs,” in *Proceedings of the 6th International Symposium on Software Metrics*, 1999.