

On the Automatic Detection and Correction of Software Architectural Defects in Object-Oriented Designs

Naouel Moha and Yann-Gaël Guéhéneuc

6th ECOOP workshop on Object-Oriented Reengineering

Glasgow, Scotland

2005/07/26



Ptidej Team – OO Programs Quality Evaluation and Enhancement using Patterns

Group of Open, Distributed Systems, Experimental Software Engineering

Department of Informatics and Operations Research

University of Montreal

© Moha and Guéhéneuc 2005



Outline

- Objective
- Terminology
 - Taxonomy, Classifications, Formalization
- Detection of Software Defects
 - Techniques, Tools
- Correction of Software Defects
 - Techniques, Tools
- Challenges



Objective

■ Our Aim

- “Formalize SAD* including antipatterns and design defects for their detection and correction in object-oriented architectures and to correct them”

■ Perspectives

- Short term: Get some feedback from the WOOR participants
- Mid term: Formalization of SAD defects and analysis of techniques
- Long term: Techniques and tools for automatic detection and correction

■ Related Work

- “Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis”, Brown *et al.*

Terminology

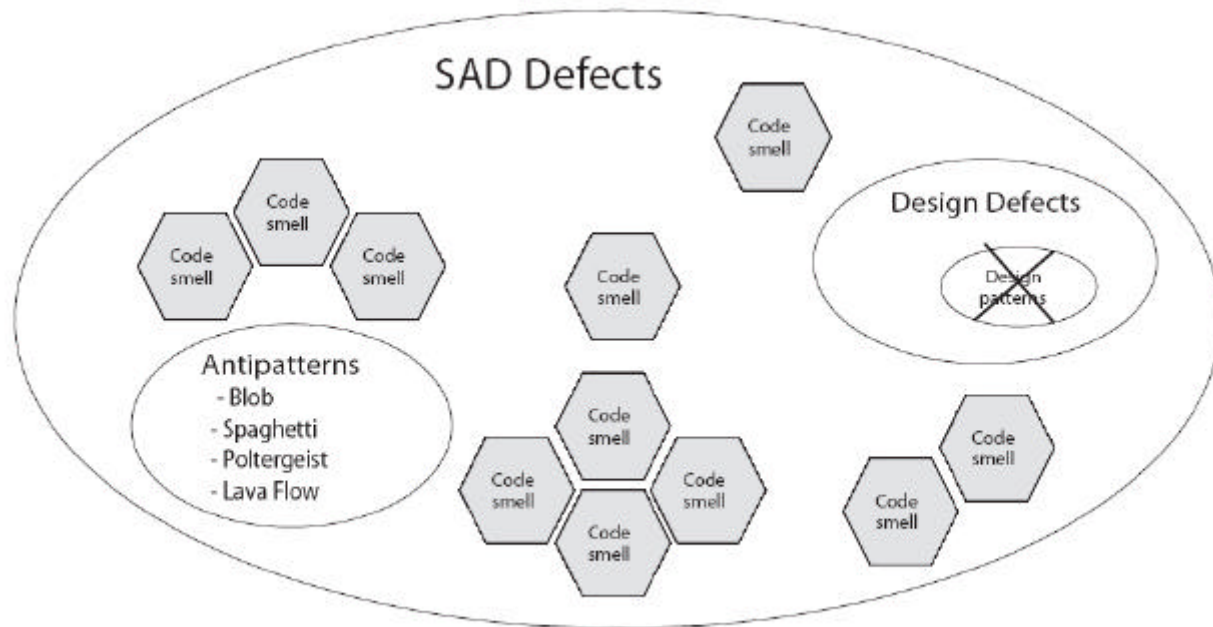
(1/3)

- Taxonomy
 - Antipatterns, design defects, code smells
- Classifications
 - Few classifications of defects at design level
- Among SAD, three categories [1]
 - Intra-classes: Internal structure of a class
 - Inter-classes: External structure of the classes (public interface) and their relationships (inheritance, association...)
 - Behavioural: Semantics of the program

Terminology

(2/3)

■ Classifications



Terminology

(3/3)

■ Formalization

- Only textual descriptions of software defects but no formal description of SAD
 - Ex: Detection of antipatterns by intuition
 - Design defects?
- Meta-models
 - PADL [2], FAMIX [3], DMP [4]
 - Right meta-model to formalise and to detect SAD defects in OO architectures?

Detection of Software Defects

■ Existing Techniques

- Reading techniques (software inspections)
- At code level: Functional or structural testing

■ Suggestions

- Extraction of code comments
- Behavioural analysis based on the sequence diagrams
- Dynamic detection during execution (assertions, pre- and post-conditions)
- Metrics (number of methods, attributes, LOC)
- Code smells: Heuristics at the code level for an automatic detection
- **Are they good?**

■ Tools

- Ex: OptimalAdvisor, IBM Structural Analysis for Java (SA4J), SmallLint
 - Help in understanding and in visualizing the structure of the code
 - Provide some basic measures (size, inheritance)
 - Highlight some problems
 - **Limited:** Do not support the detection of defects at the design level



Correction of Software Defects

■ Techniques

- Behaviour preserving and non-behaviour preserving refactoring techniques
- Semi-automatic correction

■ Our aim

- To precise, to develop, or to extend refactoring techniques
- Generic techniques applicable on any SAD

■ Tools

- Basic refactorings: package/class/method rename/move/remove, variable insert/remove/rename...



Challenges

■ Future work

- Classifying SAD
- Formalizing SAD
- Defining techniques, tools, and metrics for the automatic detection and the semi-automatic correction of SAD
- Implementing and validating these techniques and tools

References

- [1] Yann-Gaël Guéhéneuc and Hervé Albin-Amiot. Using design patterns and constraints to automate the detection and correction of inter-class design defects. In Quioyun Li, Richard Riehle, Gilda Pour, and Bertrand Meyer, editors, proceedings of the 39th conference on the Technology of Object-Oriented Languages and Systems, pages 296–305. IEEE Computer Society Press, July 2001.
Available at: www.yann-gael.gueheneuc.net/Work/Publications/.
- [2] Hervé Albin-Amiot and Yann-Gaël Guéhéneuc. Meta-modeling design patterns: Application to pattern detection and code synthesis. In Bedir Tekinerdogan, Pim Van Den Broek, Motoshi Saeki, Pavel Hruby, and Gerson Sunyé, editors, proceedings of the 1st ECOOP workshop on Automating Object-Oriented Software Development Methods. Centre for Telematics and Information Technology, University of Twente, October 2001. TR-CTIT-01-35.
Available at: www.yann-gael.gueheneuc.net/Work/Publications/.
- [3] Serge Demeyer, Stéphane Ducasse, and Sander Tichelaar. Why FAMIX and not UML? Technical report, Software Composition Group, University of Bern, 1999.
Available at: iamwww.unibe.ch/famoos/FAMIX/whyFAMIX/whyFAMIX.html.
- [4] Kim Mens, Isabel Michiels, and Roel Wuyts. Supporting software development through declaratively codified programming patterns. Elsevier Journal on Expert Systems with Applications, 23(4). Lecture Notes in Computer Science (LNCS), November 2002.