Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science 141 (2005) 117–136
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Adaptive Detection of Design Flaws

Jochen Kreimer[1, 2]

Research Group Programming Languages and Compilers
Department of Computer Science
University of Paderborn, Germany

Abstract

Criteria for software quality measurement depend on the application area. In large software systems criteria like maintainability, comprehensibility and extensibility play an important role.
My aim is to identify design flaws in software systems automatically and thus to avoid "bad" — incomprehensible, hardly expandable and changeable — program structures.
Depending on the perception and experience of the searching engineer, design flaws are interpreted in a different way. I propose to combine known methods for finding design flaws on the basis of metrics with machine learning mechanisms, such that design flaw detection is adaptable to different views.
This paper presents the underlying method, describes an analysis tool for Java programs and shows results of an initial case study.

Keywords: Design flaw, code smell, object-oriented design, software quality, refactoring, program analysis, and machine learning.

1 Introduction

The object-oriented programming paradigm promises clearly structured, re-usable and easily maintainable software. In practice, only very experienced developers achieve this.

"All data should be hidden within its class" [36] is but one of the numerous helpful pieces of advice of known mentors and successful practitioners of the

[1] Email: `jotte@uni-paderborn.de`
[2] Thank to M. Thies, P. Pfahler, U. Kastens, and the anonymous reviewers for their valuable comments.

doi:10.1016/j.entcs.2005.02.059

object-oriented paradigm that should lead to a critical review of one's own program structures.

One valuable technique to improve software quality is manual software inspection [10] [9]. It incorporates sifting source code, design and documentation. This plays an important role for quality assurance in modern agile development processes like *Extreme Programming*.

With inspection techniques, errors might be found before testing which means that they are found in early development stages. Tool support is recommended for this time consuming task. Using a tool that analyzes software automatically and repeatedly should achieve a constant high level of quality.

It is my aim to find errors in the design of software systems automatically and therefore to avoid program structures that can not easily be extended and changed.

Section 2 describes and classifies the notion of design flaws. Afterwards section 3 introduces a method for adaptive detection of design flaws. Section 4 describes the prototype tool *It's Your Code* (IYC) which implements that method. The applied program analysis techniques are outlined in section 5. The remaining sections describe results of an initial case study and provide an overview of related work, as well as a conclusion and plans for future work.

2 Design Flaws

Design flaws are program properties that point out a potentially erroneous design of a software system.

In the literature these are also referred to as *"Design Heuristics"* [36], *"Design Characteristics"* [46] or *"Bad Smells"* [13]. The authors denote design flaws normally using metaphors and explain to software developers and architects how to recognize and correct such erroneous software structures.

In addition Fowler [13] describes *refactoring* transformations which change the internal program structure, but do not alter the observable behavior of the program. These transformations lead to revised and simplified programs. Fowler's *"Bad Smells"* are design flaws that describe which program locations may get improved by *refactoring* transformations. Examples of *"Bad Smells"* are long methods, multipurpose classes, too many parameters or local variables in a method, violation of data encapsulation, overuse of delegation or misuse of inheritance.