Towards an Automated Approach for Quality Improvement in Object-Oriented Design

Radu Marinescu*

* Institute e-Austria Timişoara
radum@cs.utt.ro

Abstract

The industry is nowadays confronted with large-scale monolithic and inflexible object-oriented software. Because of their high business value, these legacy systems must be re-engineered, in order to improve the quality of their design. One of the important issues in this process is the detection and location of those design flaws, which prevent an efficient maintenance and further development of the system.

In this paper we present a metrics-based approach for detecting design problems, and assess the quality of design in an automated manner. The approach is based on a novel concept, i.e. detection strategy, which allows us to express in a measurable manner violations of design rules and heuristics. Using detection strategies we propose a new type of quality model, that does not only report quality drawbacks but also support the eventual improvement of design.

1 Detection of Design Flaws

Object-oriented programming is a basic technology, that supports quality goals like maintainability and reusability but just knowing the syntax elements of an object-oriented language or the concepts involved in the object-oriented technology is far from being sufficient to produce good software. A good object-oriented design needs design rules and practices that must be known and used. Their violation will eventually have a strong impact on the quality attributes. But how can we control and asses the usage of these rules and practices? The answer is: by measuring the design [1].

1.1 Detection Strategies

But, the main issue in working with metrics is how should we deal with measurement results? How can all those numbers help us improve the quality of our software? Most of the times a metric alone cannot help very much in answering this question and therefore metrics must be used in combination to provide relevant information. But how should we combine metrics in order to make them serve our purposes?

The main goal of the mechanism presented below is to provide the engineers with a mechanism that will allow them to work with metrics on a *more abstract level*, which is conceptually much closer to the real intentions in using metrics. The mechanism defined for this purpose is called *detection strategy*:

Definition 1.1 (Detection Strategy) *A* detection strategy *is the quantifiable expression of a rule by which design fragments that are conforming to that rule can be detected in the source code.*

A *detection strategy* is therefore a generic mechanism for analyzing a source code model using metrics. The use of metrics in the detection strategies is based on the mechanisms of *filtering* and *composition*. The role of the *filtering mechanism* is to reduce the initial data set, so that only those values are retained that present a special characteristic. This The purpose for doing that is to detect those

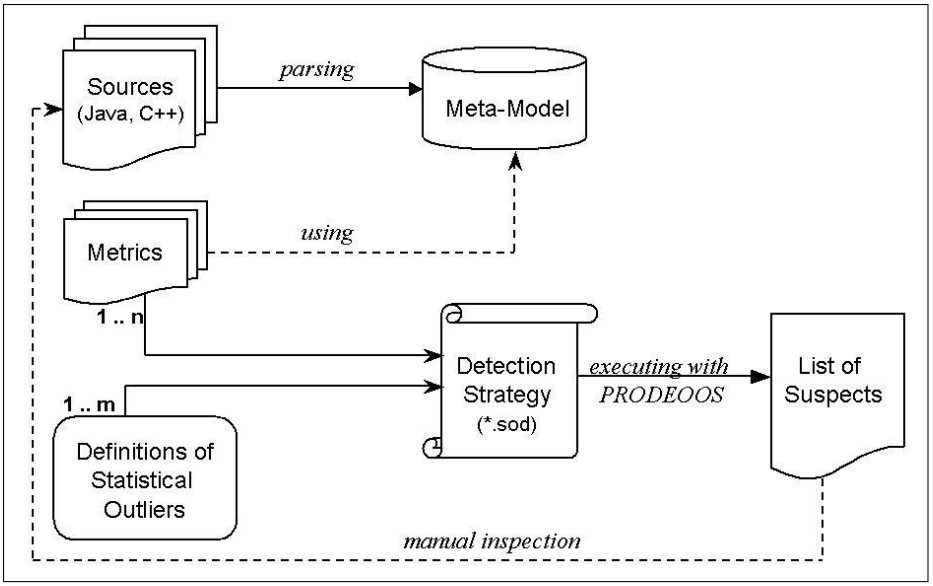

Figure 1: An overview of the automatic detection of design flaws using detection strategies.

design fragments that have special properties captured by the metric. On the other hand, the role of the *composition* mechanism is to support a correlated interpretation of *multiple result sets*.

Using detection strategies, we succeeded [4] in quantifying a large set of well-known design flaws described in the literature on different abstraction levels, from design problems affecting methods (e.g. "Feature Envy" [2]) and classes (e.g. "God Classes" [8]) up to flaws of subsystem design (e.g. "Wide Subsystem Interface" [5]) or even missing design patterns that could have been applied (e.g. "Lack of Bride" [3]).

1.2 Automatizing the Detection of Design Flaws

The detection process associated with our approach consists of the following steps (see Figure 1):

1. **Construction of the System Model.** In order to apply the metrics on a given project, we need to have all necessary design information. This is stored in a *meta-model*. The meta-model consists of information about the design entities of the system and about the existing relations among these entities. The extraction of this meta-model is a step that must be done only one time per project.

2. **Definition of Detection Strategy.** This step mainly consists in selecting a set of metrics which are appropriate for detecting a particular design characteristic (e.g. a design flaw). Metrics are defined as queries on the meta-model built in the previous step. For each metric we associate one or more statistical operators, based on which we can specify which values are the outliers of the metric. Finally, metrics are combined in a strategy using logical connectors. This step must be taken only when a new strategy is defined, independent of the number of analyzed projects.

3. **Running the Detection Strategy.** After a detection strategy is defined, we run it automatically using the PRODEOOS tool in order to get the detection results for the analyzed project. The result is a set of design entities together with the values for the different metrics that were involved in the detection strategy. This step is fully automatized.

4. **Verifying the Results.** The results obtained in the previous step must be manually inspected in order to decide if the suspects conform to the rule quantified in the detection strategy.

The last two steps of the detection process are run for every project and every detection strategy that must be checked for the analyzed project.

2 The Factor-Strategy Quality Model

Without an assessment of product quality, speed of production is meaningless. This observation has led software engineers to develop *models of quality* whose measurements can be combined with those of productivity. Quality models are built in a decompositional manner, whereby the best known model is the *Factor-Criteria-Metric*[6].

The main difficulty of these classical approaches, can be synthesized as follows: the possibility to identify the real causes of quality weaknesses as perceived from the outside (e.g. poor maintainability, low reuse level) is strongly hampered by the fact that the metrics level in quality models is too fine-grained to allow an understanding of the real design problems, which consequently hinders the proper redesign decisions, for a long-term increase of quality.

Therefore, we define a new quality model, i.e. *the Factor-Strategy Quality Model* in which quality factors are expressed (in the sense of decomposition) in a set of quantifiable rules that identify violations of design principles, rules and heuristics. Of course, these quantifiable rules are given by the suite of detection strategies defined in the previous section. Thus, instead of communicating directly with a "bunch" of numbers, which has a low relevance level, in our approach quality is expressed and evaluated in terms of an explicit *"knowledge-box of object-oriented design"* containing the quantified expressions of the good-style design rules for the object-oriented paradigm.

Definition 2.1 (Factor-Strategy Quality Model (FS)) *A model used for the assessment of software quality that decomposes quality in a set of high-level quality factors and maps each factor to a set of strategies that quantify deviations from rules of good design is called a* Factor-Strategy Quality Model.

In the previous definition we distinguish the following components of a Factor-Strategy quality model:

1. **Quality Factors** have precisely the same meaning as in the FCM models, i.e. high-level attributes used as an expression of quality as perceived by the user.

2. **Strategies** are the means used to detect and quantify the flaws at the design and implementation level that negatively rebound upon quality factors. A strategy encapsulates a metrics-based mechanism for finding deviations from a particular rule of good design. The strategy in a Factor-Strategy quality model is in fact an instance of the *detection strategy* concept.

At first sight (see Figure 2) it becomes obvious that FS models still use a decompositional approach, but after decomposing quality in factors, these factors are not anymore associated directly with a bunch of numbers, which proved to be of a low relevance for an engineer. Instead, quality factors are now expressed and evaluated in terms of detection strategies, which are the quantified expressions of the good-style design rules for the object-oriented paradigm.

Therefore we may state in more abstract terms that *in a Factor-Strategy model, quality is expressed in terms of principles, rules and guidelines of a programming paradigm.* The set of detection strategies defined in the context of a FS quality model encapsulate therefore the *knowledge-box of good design* for the given paradigm. The larger the knowledge-box, the more accurate the quality assessment is.

The knowledge-box as such is indispensable for any quality model. Although not visible at first sight, it is also present in the FCM models. The knowledge-box is not obvious in the FCM approach because of its *implicit character*. The entire knowledge about the programming paradigm is captured in the association of criteria with metrics. Using a metaphor, we might say that the knowledge-box is behind the arrows that link criteria with metrics. Of course, the metrics are also paradigm-specific, but the design and programming principles that the metrics reflect are implicit. They may be inferred from the construction of the model, but they are not explicit. Thus, the novelty of the FS approach resides in the *explicitness* of the principles and rules that support the quality of a system.

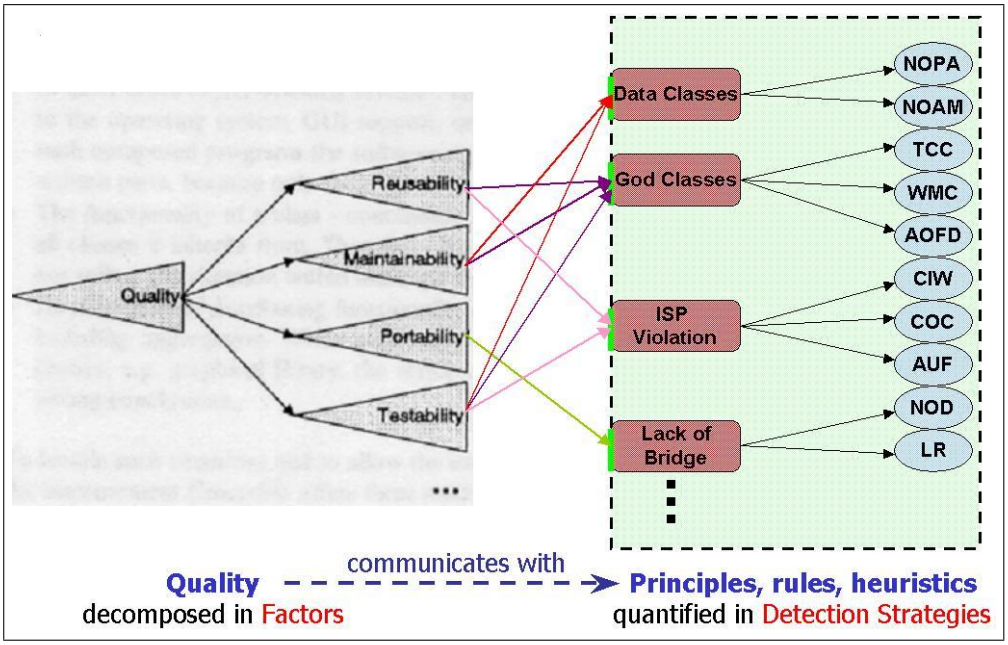

Figure 2: A Factor-Strategy quality model.

This explicitness is supported by the use of detection strategies as quantifications of these rules and principles.

References

[1] T. DeMarco. *Controlling Software Projects; Management, Measurement and Estimation*. Yourdon Press, New Jersey, 1982.

[2] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[4] R. Marinescu. *Measurement and Quality in Object-Oriented Design*. Ph.D. Thesis, Timişoara, 2002.

[5] R.C. Martin. *Design Principles and Patterns*. Object Mentor, http://www.objectmentor.com, 2000.

[6] J.A. McCall, P.G. Richards, and G.F. Walters. *Factors in Software Quality, Volume I*. NTIS AD/A-049 014, NTIS Springfield, VA, 1977.

[7] W. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[8] A.J. Riel. *Object-Oriented Design Heuristics*. Addison-Wesley, 1996.