

Measurement and Quality in Object-Oriented Design

Radu Marinescu
LOOSE Research Group
"Politehnica" University of Timișoara
Bvd. V. Pârvan 2, 300223 Timișoara (Romania)
radum@cs.utt.ro

Abstract

In order to support the maintenance of object-oriented software systems, the quality of their design must be evaluated using adequate quantification means. In spite of the current extensive use of metrics, if used in isolation, metrics are oftentimes too fine grained to quantify comprehensively an investigated aspect of the design. To help the software engineer detect and localize design problems, the novel detection strategy mechanism is defined so that deviations from good-design principles and heuristics are quantified in form of metrics-based rules. Using detection strategies an engineer can directly localize classes or methods affected by a particular design flaw (e.g. God Class), rather than having to infer the real design problem from a large set of abnormal metric values. In order to reach the ultimate goal of bridging the gap between qualitative and quantitative statements about design, the dissertation proposes a novel type of quality model, called Factor-Strategy. In contrast to traditional quality models that express the goodness of design in terms of a set of metrics, this novel model relates explicitly the quality of a design to its conformance with a set of essential principles, rules and heuristics, which are quantified using detection strategies.

1 Introduction

There is no perfect software design. Like all human activities, the process of designing software is error prone and object-oriented design makes no exception. The flaws of the design structure have a strong negative impact on quality attributes such as flexibility or maintainability [4]. Thus, the identification and detection of these design problems is essential for the evaluation and improvement of software quality.

The fact that the software industry is nowadays confronted with a big number of large-scale legacy systems written in an object-oriented language, which are monolithic, inflexible and hard to maintain shows that 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. Thus, on one hand, a good object-oriented design needs design rules and heuristics [6], [18] that must be

known and used. On the other hand, as De Marco pointed out [2], in order to control the quality of a design we need proper quantification means. But is it possible to express these "good-design" rules in a quantifiable manner?

2 Problem Statement

The previous experiences of applying object-oriented design metrics for the assessment and improvement of design quality in object-oriented systems are encouraging [3], [10], [7] but, the use of metrics as it is now raises a set of new problems, summarized below:

- The interpretation of individual measurements is too fine-grained if metrics are used in isolation; this reduces the applicability and relevance of metrics usage. Thus, in most cases individual measurements do not provide relevant clues regarding the *cause* of a problem. In other words, a metric value may indicate an anomaly in the code but it leaves the engineer mostly clueless concerning the ultimate cause of the anomaly.
- There is a lack of relevant feedback link in quality models. In other words, we measure, we set thresholds, we identify suspects, but the metric by itself does not provide enough information for a transformation of the code that would improve quality. Thus, the developer is provided only with the problem and he or she must still empirically find the real cause and eventually look for a way to improve the design.

In conclusion, there is a major gap between the things that we measure and the issues that have an important quality impact at the design level.

3 The Thesis

The gap between qualitative and quantitative statements, concerning object-oriented software design can be bridged using higher-level, goal-driven methods for measurement interpretation.

The goal of the dissertation was to develop methods and techniques that provide a relevant interpretation of measurement results applied to the investigation of object-oriented software design.

4 Detection Strategy

As already mentioned, a metric does not provide in itself enough information for making a decision for a code transformation that would improve quality. In this context, this work introduced a new mechanism, named *detection strategy*, for increasing the relevance and usability of metrics in object-oriented design by providing a higher-level (more abstract level) means for interpreting measurement results. A detection strategy was defined in the work [11] as: "the quantifiable expression of a rule by which design fragments that are conforming to that rule can be identified in the source-code".

The main goal of a strategy is to provide the engineer with a mechanism that will allow him or her to work with metrics on a more abstract level, which is conceptually much closer to his real intentions in using metrics. Such rules may refer both to design recovery (*understanding of design*) and the identification of design flaws (*evaluation of design*).

4.1 Detection Strategy. An Example

Let's assume that we want to find the poor encapsulated classes *i.e.*, those classes that exhibit their data in the interface. For this we use two metrics: the first one to count the number of public attributes (NOPA) and the other one to count the number of accessor methods (NOAM) [10]. We decide that the classes we want to find are those with the most public data from the project, but that they should not have less than 3 public attributes or 5 accessor-methods. Therefore, the detection strategy can be expressed as follows¹:

```
PoorEncapsulatedClasses :=
  (NOPA, HigherThan(3) and NOPA, TopValues(10%))
or (NOAM, HigherThan(5) and NOAM, TopValues(10%))
```

As you might have already noticed, the most sensitive part in a detection strategy is the selection of threshold values. The issue is addressed in detail in the dissertation [11] and in [16].

4.2 Comprehensive Catalogue of Detection Strategies

In order to support the assessment of design quality, the dissertation defined a suite of 15 detection strategies, that either quantify deviations from different good-design rules found in the literature [14, 18] or "bad smells" of code [4]. The flaws are classified in conformity with the design entities that they affect *i.e.*, methods, classes and subsystems. In addition to these we introduced a fourth category: *micro-design flaws*. Design flaws

¹The sequence is written in SOD, a script language defined as part of the ProDeOOS toolkit described in the dissertation

that fall in this category affect not only a class, but a cluster of classes (or subsystems). In this category we included those cases where a particular design pattern [5] should have been applied, but the pattern solution was ignored. Figure 1 summarizes

| Category | Name | Impact on: | | | |
|------------------------------------|-----------------|------------|-------|--------|--------|
| | | COUPL | COHES | COMPLX | ENCAPS |
| Class | GodClass | X | X | X | |
| | DataClass | | | | X |
| | ShotgunSurgery | X | | | |
| | RefusedBequest | | X | X | |
| | ISPViolation | X | | | |
| Method | GodMethod | | | X | |
| | FeatureEnvy | X | X | | X |
| | TemporaryField | | | X | |
| Subsystem | GodPackage | X | | | |
| | MisplacedClass | | X | | |
| Micro-Design (missing patterns) | LackOfBridge | X | | X | |
| | LackOfStrategy | | X | X | |
| | LackOfState | | | X | |
| | LackOfSingleton | X | | | X |
| | LackOfFacade | X | | | |

Figure 1. Overview of design flaws

these detection strategies. Each of the design flaws is put in relation with the four criteria of good design identified by Coad and Yourdeon in [1]. These are: *low coupling* (COUPL), *high cohesion* (COHES), *manageable complexity* (COMPLX) and *proper data abstraction* (ENCAPS).

5 Factor-Strategy Quality Model

The ultimate goal of this dissertation was to demonstrate that the gap between qualitative and quantitative statements can be bridged. Therefore we focused our attention inevitably on the issue of quality models. The first discovery was a general drawback of classical approaches [15] which can be synthesized as follows: the ability to identify the real causes of quality weaknesses as perceived from the outside (*e.g.*, poor maintainability, low reuse level) is severely hampered by the fact that the metrics level in *Factor-Criteria-Metrics* (FCM) 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.

As a consequence, the *Factor-Strategy* (FS) quality model was defined, being a novel approach 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 (see Figure 2²). The "quantifiable rules" are expressed by the suite of detection strategies defined earlier in this work (see Section 4.2).

²The acronyms that appear in the ovals on the right side of Figure 2 represent metrics, and the arrows show which metric appears in which detection strategy (rectangles). The concrete metrics are not relevant for the understanding of this picture

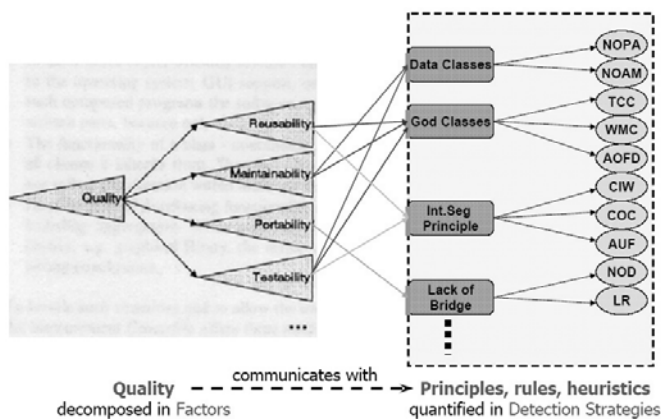


Figure 2. Factor-Strategy model. The concept

The set of detection strategies used in the context of a *Factor-Strategy* quality model can be regarded as a “*knowledge-box*” of good design for the given design 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 “classical” FCM models. The knowledge-box is not obvious in the FCM approach because of its *implicit character*, while it becomes *explicit* in the FS model.

The *Factor-Strategy* approach has two major improvements over the traditional approaches:

- The *construction* of the quality model is easier because the quality of the design is naturally and explicitly linked to the principles and good-style rules of object-oriented design. This approach is in contrast with the classical Factor-Criteria-Metric approach, where in spite of the decomposition of external quality factors into measurable criteria, quality is eventually linked to metrics in a way that is less intuitive and natural.
- The *interpretation* of the strategy-driven quality model occurs at a higher abstraction level *i.e.*, the level of design principles, and therefore it leads to a direct identification of the real causes of quality flaws, as they are reflected in flaws at the design level. As we pointed out earlier, in this new approach quality is expressed and evaluated in terms of an explicit knowledge-box of object-oriented design.

Besides the improvement of the quality assessment process, the detection strategies used in the context of a Factor-Strategy quality model proved to have a further applicability, at the conceptual level: for the first time a quality factor could be described in a concrete and sharp manner with respect to a given programming paradigm. This is achieved by describing the quality factors in terms of the detection strategies that capture design problems that affect the quality factor, within the given paradigm.

6 Contributions Summary

The approach presented in this work brings a number of essential contributions to the field of quality assessment in object-oriented design based on software metrics. These contributions are summarized below as follows:

- **Definition of the *detection strategy* concept** as a proper means for a higher-level measurement interpretation. Detection strategies are a means for defining metrics-based rules. They offer an encapsulation of metrics in a higher-level construct that permits a more meaningful interpretation and usage. For the first time, detection strategies become a mechanism for “articulating” (expressing, quantifying) rules related to the structure of code and design in terms of metrics.
- **A method usable in practice for quantifying informal code- and design-related rules.** The dissertation does not only define the mechanism to objectify and quantify such rules, but it does also provide a methodology for moving from informal descriptions of rules to detection strategies.
- **A suite of detection strategies for the identification of well-known design flaws.** For the first time descriptions of design problems like “Feature Envy” [4] or “God Classes” [18] are quantifiable and detectable. In addition to that, the suite also contains detection strategies for identifying places in the source code where a particular design pattern should have been applied.
- **Definition of a quality model based on detection strategies.** Although the proposed model is also decompositional one, yet for the first time in this model quality communicates with the principles, rules and heuristics of good object-oriented design – objectified by the different detection strategies – instead of being mapped to “pure numbers” (measurement results). The main advantage of this new quality model is that it leads to a direct identification of the real causes of quality flaws, as they are reflected in flaws at the design level.
- **Strong tool support for the high level of automatization and scalability** of the approach. The tool support is covering all the methods and techniques presented in this work from the definition and execution of new metrics and detection strategies, up to the definition of complex quality models based on detection strategies. The tool support has a high-level of language-independency, as it works on a unified meta-model for object-oriented languages. The provided tool support makes the approach scalable, and this is proved by the fact that the tools were used for analyzing real-world systems containing up to 1 million LOC.
- **Concrete evaluation of the concepts, methods and techniques** defined and presented in this thesis, based on two versions of an industrial case-study and a suite of smaller

research and university projects. We have shown in the dissertation, based on the case studies that the *Factor-Strategy* quality model is usable in practice and provides the engineer with information that is not only relevant for quality assessment, but also for the further improvement of the system's design.

7 Scientific and Practical Impact of the Thesis

The relevance of the concepts introduced and developed by this thesis is emphasized also by the publications that resulted from it. Thus, apart from the thesis itself [11], several papers [8, 10, 12, 13, 16] were accepted in the mainstream conferences of the maintenance and reengineering community. Currently, a book – entitled *Object-Oriented Metrics in Practice* – containing an enhanced version of this work is in print at *Springer Verlag*.

In the years following the defense of the thesis, the concepts introduced here inspired the research of other Ph.D. students: Rațiu and Gîrba [17] applied the detection strategy concept for finding design problems that are revealed by the history of the system; while Trifu [19] defines *correction strategies* as a mean to address by refactorings the design problems which are detected using the suite of detection strategies defined in this dissertation.

Concerning the practical impact of the thesis, it is worth to mention that the *detection strategy* concept introduced by this dissertation, together with a suite of metrics used for measuring inheritance-based coupling [9], were integrated in a well-known family of commercial CASE tools *i.e.*, Borland's Together Control Center product family.

References

- [1] P. Coad and E. Yourdon. *Object-Oriented Design*. Prentice Hall, London, 2 edition, 1991.
- [2] T. DeMarco. *Controlling Software Projects; Management, Measurement and Estimation*. Yourdan Press, New Jersey, 1982.
- [3] K. Erni. *Anwendung multipler Metriken bei der Entwicklung objektorientierter Frameworks*. PhD thesis, Universität Karlsruhe, Mnster, 1996.
- [4] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.
- [5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [6] R.E. Johnson and B. Foote. Designing reuseable classes. *Journal of Object-Oriented Programming*, 1(2):22–35, June 1988.
- [7] M. Lanza and S. Ducasse. A Categorization of Classes based on the Visualization of their Internal Structure: the Class Blueprint. In *OOPSLA 2001 proceedings*, 2001.
- [8] Radu Marinescu. Using object-oriented metrics for automatic design flaws in large scale systems. In Serge Demeyer and Jan Bosch, editors, *Object-Oriented Technology (ECOOP '98 Workshop Reader)*, volume 1543 of *LNCS*, pages 252–253. Springer-Verlag, 1998.
- [9] Radu Marinescu. A Multi-Layered System of Metrics for the Measurement of Reuse by Inheritance. In *Proceedings of TOOLS Asia 1999*, pages 142–153. IEEE Computer Society, 1999.
- [10] Radu Marinescu. Detecting Design Flaws via Metrics in Object-Oriented Systems. In *Proceedings of TOOLS USA 2001*, pages 103–116. IEEE Computer Society, 2001.
- [11] Radu Marinescu. *Measurement and Quality in Object-Oriented Design*. Ph.D. thesis, Department of Computer Science, "Politehnica" University of Timișoara, 2002.
- [12] Radu Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *Proceedings of ICSM '04 (International Conference on Software Maintenance)*, pages 350–359. IEEE Computer Society Press, 2004.
- [13] Radu Marinescu and Daniel Ratiu. Quantifying the quality of object-oriented design: The factor-strategy model. In *Proceedings of WCRE 2004 (Working Conference on Reverse Engineering)*, pages 192–201, 2004.
- [14] R.C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall: 1st Edition, 2002.
- [15] 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.
- [16] Petru Florin Mihancea and Radu Marinescu. Towards the optimization of automatic detection of design flaws in object-oriented software systems. In *Proceedings of CSMR 2004 (European Conference on Software Maintenance and Reengineering)*, pages 92–101, 2005.
- [17] Daniel Rațiu, Stéphane Ducasse, Tudor Gîrba, and Radu Marinescu. Using history information to improve design flaws detection. In *Proceedings of CSMR 2004 (European Conference on Software Maintenance and Reengineering)*, pages 223–232, 2004.
- [18] A.J. Riel. *Object-Oriented Design Heuristics*. Addison-Wesley, 1996.
- [19] Adrian Trifu, Olaf Seng, and Thomas Genssler. Automated design flaw correction in object-oriented systems. In *Proceedings of CSMR 2004 (European Conference on Software Maintenance and Reengineering)*, pages 174–183, 2004.