Abstract

Software refactoring - improving the internal structure of the software without changing its observable behavior - is an important activity towards avoiding software quality decay. Key to this activity is the identification of portions of the source code that offers opportunities for refactoring -- the so called bad smells. The goal of this position paper is the discussion of an approach to help on the detection of code bad smells through source code metrics and the results obtained from its use. In this discussion, it is also important to bring new elements that might be affected through a refactoring sequence as, for example, structural testing requirements that can be used in the future as a new metric to detect refactoring opportunities.

1. Introduction

According to Fowler [7], refactoring increases software comprehension and, as a consequence, makes it easier to be modified. Refactoring relies on adaptations of the original source code to increase its flexibility and reuse [14, 15]. Although it is possible to refactor manually, tool support is considered crucial. Today, a wide range of tools and methodologies is available to automate various aspects of refactoring [8, 12, 16]. The focus of available tools is on applying a refactoring or restructuring upon request of the user. There is less support available for detecting where, why, and when a refactoring operation can be applied [12]. This position paper discusses the use of the methodology proposed in [3, 4] as well as future directions to apply source code metrics to detect refactoring opportunities. The rest of this position paper is organized as follows: Section 2 presents the refactoring process; Section 3 describes the proposed methodology in order to show the context in which the study has been developed, presents an evaluation of structural testing requirements in refactoring activities and some open questions related to the detection of refactoring. Section 4 presents the conclusions.

2. Refactoring Process

The refactoring process is composed by the following steps: (a) Detection: identify portions of code with refactoring opportunities; (b) Cost-benefit analysis: determine what are the main motivations and benefits for refactoring; (c) Evaluation arrangements: prepare the code for refactoring; (d) Execution: execute refactoring activities; (e) Evaluation: verify the preservation of program behavior.

The methodology proposed in [7] aims to detect refactoring opportunities [6] - step (a) - and its use. This step is not supported by currently available software refactoring tools [12, 13].

3. Methodology Proposed

The first step towards establishing a methodology to help on the detection of software refactoring opportunities using source code metrics was already presented in [3, 4], as well as the relationships between some of the refactorings listed in Fowler's book [7] and source code metrics. The hypothesis was that these metrics can eventually be used to help on refactoring activities as well as to analyze the refactoring results. But there are still open questions remaining to establish refactoring activities more predictable and less dependent on human intuition through the use of measurement and quantitative data.

Figure 2 shows the methodology proposed in [3]. It is based in the assumption that metrics suites can be created to identify refactoring opportunities in a oriented object source code program. The methodology consists of: (a) measurement of the source code program; (b) use of the measurements results to point out bad smells; (c) application of cognitive analysis to evaluate the necessity of refactoring(s) as well as the presence of false positives and negatives indications of bad smells; (d) use of one or more refactorings to remove the bad smells, as indicated by Fowler [7].

The methodology shows that it is possible to associate measurement results with bad smells to improve code quality [6]. The main advantage of this approach is the reduction of the amount of source code to be analyzed by a human expert [9]. However, as presented in Figure 2, the approach can produce false negatives and positives indications of bad smells. False positives indicate bad smells that do not really exist; they should be eliminated by cognitive analysis. False negatives are bad smells that exist in the source code but are not detected by the metrics. They are difficult identify by cognitive analysis, because the expert does not have an indication of their location. It is assumed that the reduction of source code analysis made possible by the proposed methodology...
To achieve a relationship between metrics, refactorings [5] and bad smells the following approaches were considered: (a) A top-down approach based on Goal Question Metric (GQM) paradigm [1]; (b) A bottom-up approach based on empirical analysis. The first approach made an analytical discovery of metrics used in bad smells evaluation. The second approach is empirical and used results from large metrics suites in order to verify its relationships with refactorings and bad smells. Those approaches are coherent with the methodology proposed by Mendonça and Basili to create and improve measurement frameworks in software engineering [10,11].

The authors are also investigating the possibility of using structural testing requirements as a new metric to detect refactoring opportunities based on empirical case study using Jabuti [17]. Some requirements had a considerable decreasing through refactoring, steps, which can be investigated to verify the possibility using it as a new metric in the methodology proposed in [3].

The following open questions are so far identified: (a) Can test requirements be used as a new metric to detect refactoring opportunities? (b) How to reduce even more the incidence of false positives and negatives? (c) How to formalize and discover the smallest and most efficient set of metrics to detect refactoring opportunities?

4. Conclusion
Refactoring has definitively been incorporated to the software evolution process [2], despite the remaining of a number of open questions. Although commercial refactoring tools are available, there are still a lot of open issues that remain to be solved. The topics discussed in this position paper have been used as references for research by the authors.

5. References