Towards a Framework for Automatic Correction of Anti-patterns

Rodrigo Morales, SWAT, Polytechnique Montréal, Canada; rodrigo.morales@polymtl.ca

***Abstract*—One of the biggest concerns in software maintenance is design quality; poor design hinders software maintenance and evolution. One way to improve design quality is to detect and correct anti-patterns (i.e., poor solutions to design and implementation problems), for example through refactorings. There are several approaches to detect anti-patterns, that rely on metrics and structural properties. However, finding a specific solution to remove anti-patterns is a challenging task as candidate refactorings can be conflicting and their number very large, making it costly. Hence, development teams often have to prioritize the refactorings to be applied on a system. In addition to this, refactoring is risky, since non-experienced developers can change the behaviour of a system, without a comprehensive test suite. Therefore, there is a need for tools that can automatically remove anti-patterns. We will apply meta-heuristics to propose a technique for automated refactoring that improves design quality.**

I. Background

Software maintenance is defined as the process of modifying a software system in order to add new features, correct faults or improve functionality. In previous studies [1], the cost of software maintenance has been estimated to more than 70% of the total cost of a software. Thus, researchers have focus their effort on studying the quality of software systems and proposed metrics and methodologies to assess their condition. Some indicators of poor quality are anti-patterns [2], which depict bad design-choices that makes it hard to understand, modify and extend a software. To remove anti-patterns, practitioners perform refactoring [3, 4], which is the process of reordering, and rewriting existing code, without changing its original behaviour. However, manually refactoring is an error-prone task as it is possible to introduce defects when applied without a set of comprehensive test cases. Multiple anti-patterns detection techniques have been proposed so far [5, 6], but to the best of our knowledge, there is no effective automatic correction approach that can free developers from this difficult and time-consuming task.

II. Research Objectives

1. Perform a qualitative and quantitative study of refactorings that are applied during the development of a software system; what kind of refactorings and to what extent. To support automating anti-pattern correction, we first need to improve our understanding of how and when do developers apply refactoring, and to which extent. Through *code review*, which is the practice of having other team members critique changes to a software system, we can obtain valuable information about the development process. For the quantitative study of source code reviews, in [7], we set out to find if there is any correlation between code review and anti-patterns, using the last one as a proxy for design quality in three open-source projects. We found that components with low review coverage or low participation are more likely to present anti-patterns than components with highly-active code review practices. For the qualitative study of source code reviews, we plan to perform surveys with developers to get a better understanding of code review activities and refactoring.

2. Develop a search-based approach for the automatic refactoring of software systems. Using structural, and lexical information we can define rules to specify and detect Anti-patterns [5]. Once we have assess the level of design-defectness, i.e., an heuristic providing the design quality of the source code in a system, the next step consists of refactoring the code; this is not trivial if we consider that the number of anti-patterns can be extremely large, and normally, there is more than one candidate refactorings for removing any anti-pattern. Hence, we propose to implement a *search-based* approach [8] for the correction step due to the following reasons: 1) there is no formal approach 2) we can explore the space of feasible solutions in a reasonable time frame. To select the best heuristic to solve our problem, we explore and compare the different meta-heuristics approaches, e.g., Genetic Algorithm, Hill-Climbing, Simulated annealing, etc. We will leverage the knowledge of developers to guide our algorithm to select the best sequence of refactorings that improves the most design quality.

3. Implement our proposed approach into an Eclipse plug-in[1] to help developers to refactor their code on the fly during development and maintenance activities. We will implement our search-based refactoring approach into an eclipse plug-in. To assess the benefits of our proposed plug-in we will perform a series of usability studies. We will also compare our tool with other refactoring approaches.

III. Expected contributions

The main contributions our research can be summarized as follows:

- We expand the knowledge of source code review and the relationship with design defects.
- We propose a novel automated refactoring search-based approach to improve software quality that prioritizes the refactorings that are targeted by developers.
- We provide practitioners an Eclipse plug-in that can be used to improve the design quality of their projects.

[1] http://www.eclipse.org

REFERENCES

[1] R. S. Pressman and W. S. Jawadekar, *Software engineering - A Practitioner's Approach*, 5^{th} ed. McGraw-Hill Higher Education, 2001.

[2] W. J. Brown, R. C. Malveau, W. H. Brown, H. W. McCormick III, and T. J. Mowbray, *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*, 1^{st} ed. John Wiley and Sons, March 1998.

[3] W. F. Opdyke, "Refactoring object-oriented frameworks," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 1992.

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

[5] N. Moha, Y.-G. Gueheneuc, L. Duchien, and A. Le Meur, "Decor: A method for the specification and detection of code and design smells," *Software Engineering, IEEE Transactions on*, vol. 36, no. 1, pp. 20–36, 2010.

[6] F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, and H. Sahraoui, "Bdtex: A gqm-based bayesian approach for the detection of antipatterns," *J. Syst. Softw.*, vol. 84, no. 4, pp. 559–572, Apr. 2011.

[7] R. Morales, S. McIntosh, and F. Khomh, "Do code review practices impact design quality? a case study of the qt, vtk, and itk projects," in *22nd IEEE Int'l Conference on Software Analysis, Evolution, and Reengineering*, Submitted.

[8] M. Harman and J. Clark, "Metrics are fitness functions too," in *Software Metrics. Proc. 10th Int'l on*, 2004, pp. 58–69.