Chapter 7

Evaluating the Refactoring Index Using Entropy Approach

Rajni Sehgal
Amity University, India

Deepti Mehrotra
Amity University, India

ABSTRACT

Software often carries the structural deficiencies that make it hard to understand, change, or test; these deficiencies are categorized as a code smell. This code smell affects the performance of software adversely, thereby increase need of maintainability. Refactoring of code helps in reducing the code smell. But refactoring is an expensive process and hence identifies which and how much code need to refactor a challenging task and is termed as refactoring index. In this chapter, entropy approach is proposed to measure the refactoring index of 20 open source software programs. Refactoring index level is given to identify the critical project which urgently required refactoring in order to improve the quality of the project.

INTRODUCTION

The quality of software depends on the quality of design and source code. Poor architectural design and implementation plan may introduce the design flaws in a program which can stop the fault-free execution of the code. According to Fowler (Fowler, M., & Beck, K. 1999). Design flaws appear at the design and code level is termed as “code smell”. Code smells does not stop the functioning of the program,
because they are not bugs instead they slows down the development of any software. Induction of the code smell in software reduce the strength of architecture of any software, may lead to failure of software shortly. Although code smells are not bugs, detecting code smell reduce the testing and maintenance effort. Inspection is one of the static techniques of testing to detect the code smell during the verification of the code. During the software inspection, the potentially problematic areas are examined such as the program code, the design, and the software documentation. During the initial stage of development of the software, several problems due to an inefficient analysis of design can introduce the code smell in the software system.

Object-oriented software whose KLOC is large in number is difficult to maintain because of their architectural design. Quality of the source code is also degraded due to the poor architectural design. An established concept of code smells can classify shortcomings in object-oriented software architecture which is directly related to Antipattern present in the system design (Mansoor, U 2013). In an object-oriented software violation of design principles such as data abstraction, encapsulation, modularity, and hierarchy may affect the design of a class and increase the probability of induction of code smells in code. This may cause several problems in the later stage of software development. Once the code smell introduces in the code, it can only be cured by applying the suitable refactoring approach. Refactoring is improving the internal structure of the code without altering the functionality of any program. Planning a refractory strategy to enhance the quality of the code is an expensive and time-consuming process. Applying appropriate refactoring strategy is a multicriteria problem like it is dependent on the criticality of the project, and criticality of its constituent components. Taking a decision, which project requires more Refactoring and how to do the refactoring depends on the number and types of code smell present in the code. Different code smells have a different level of impact on the software system and hence estimating their relative importance need to be identified.

In this paper, a framework is proposed to evaluate how much refactoring termed as a refactoring index (R.I) is needed for given software. To identify which project requires at most priority towards the refactoring, level of refactoring is given in this paper, so that project with the highest refactoring index should be refactor first. Analytical approach entropy is applied to find out the disorder to predict the level of refactoring is necessary for the provided software. The entropy represents the countable information and has been suitably used in various domains to measure the relative importance of a variable on another variable (Hafezalkotob, et.al. 2016). This study is done on the open dataset repository Landfill where 20 software and their respective code smells are given by Palomba, F (2015) where he contributed this repository for five different code smell which is identified by 20 open source
Teaching Model-Driven Engineering in a Master's Program: Three Editions on a PBL-Based Experience
www.igi-global.com/chapter/teaching-model-driven-engineering-in-a-masters-program/219980?camid=4v1a