Chapter 39
Using Design of Experiments to Analyze Open Source Software Metrics for Change Impact Estimation

Miloud Dahane
https://orcid.org/0000-0002-1754-6005
Université Oran1, Oran, Algeria

Mourad Bouneffa
Université du Littoral Côte d'Opale, Dunkirk, France

Mustapha Kamel Abdi
Université Oran1, Oran, Algeria

Adeel Ahmad
Laboratoire d'Informatique Signal et Image de la Côte d'Opale, Calais, France

Henri Basson
Université du Littoral Côte d'Opale, Dunkirk, France

ABSTRACT

Software evolution control mostly relies on the better structure of the inherent software artifacts and the evaluation of different qualitative factors like maintainability. The attributes of changeability are commonly used to measure the capability of the software to change with minimal side effects. This article describes the use of the design of experiments method to evaluate the influence of variations of software metrics on the change impact in developed software. The coupling metrics are considered to analyze their degree of contribution to cause a change impact. The data from participant software metrics are expressed in the form of mathematical models. These models are then validated on different versions of software to estimate the correlation of coupling metrics with the change impact. The proposed approach is evaluated with the help of a set of experiences which are conducted using statistical analysis tools. It may serve as a measurement tool to qualify the significant indicators that can be included in a Software Maintenance dashboard.

DOI: 10.4018/978-1-7998-9158-1.ch039

1. INTRODUCTION

The software maintenance or evolution is an essential step in the software life cycle. Several works have, for many years, highlighted the importance of maintenance and/or evolution of the software and have established several taxonomies (Gasmallah et al., 2016). The oldest have allowed to consider three kinds of maintenance (Swanson, 1976): corrective maintenance leading to remove the residual errors or faults; adaptive maintenance that consists of adapting the software to changes affecting both its technical or managerial environments like the evolution of the deployment infrastructures or the change of some regulation policies, etc. The perfective maintenance includes changes intended to improve the software by introducing new features or improving some quality criteria, etc. In (Chapin, 2000), the author introduces a new maintenance type called preventive maintenance as the changes leading to make the software more able to evolve and then to change. In other words, the changes induced by this last kind of maintenance makes it possible to improve the maintainability criterion of the software without affecting the functionalities or performance of such a software.

In this work, we do not intentionally make any difference between maintenance and evolution, although these two concepts have been subjects to numerous comparisons. Maintenance is often seen as an engineering process in which academic research has delineated and characterized the stages, the activities to be carried out, the actors and resources to be implemented, etc. In other words, the concept of maintenance is the result of a kind of morphism between the industrial world and the software development activity. The term software evolution is the result of works (Lehman & Ramil, 2002.) aimed at understanding the phenomenon of change affecting the software artifacts. For example, Lehmann's laws of evolution attempt to explain how the software evolves during many years until becoming obsolete. These laws are the result of empirical studies conducted over several years on software used on the real world. In the same way, Benett and Rajlich drew up (Benett & Rajlich, 2000) a software life cycle designed not as a means to develop the software but as an explanation of how software is produced, evolved, maintained, and eventually removed from the information system.

As far as we are concerned, we indifferently use the terms maintenance or evolution. What we are particularly interested in, is the notion of the software change and more specifically the Change Impact Analysis (Abdi et al., 2009; Sun et al., 2012). It is clear that one of the major concerns of the software development stakeholders is to best control the change process and then the software maintenance/evolution one. In fact, an uncontrolled change can quickly lead to a slippage in terms of project's cost and time. During more than four decades, many works have addressed this problem making the maintenance the most important cost factor of all the software development process (Folmer & Bosch, 2008; Gupta et al., 2008). In addition, according to ISO 250010 Model (ISO, 2011) (José et al., 2014), maintainability or scalability is one of the main components of the software quality.

In this work, we are interested in estimating the impact of the software change but from a quantitative point of view. In other words, we try to produce a model based on mathematical equations allowing us to estimate, in a way, the cost of the change of a given software. It reflects the impact of change in terms of the lines of code to be further modified, as an effort required to adapt the change. Indeed, a modification in a software development results in changes made in the source code in order to improve or correct its operation. These changes include any change affecting any element of the software (variable, method, or class). This can be, for example, deleting a variable, changing the scope of a method, or moving the link between a class and its super class, etc. A change can have dramatic and unexpected effects on the rest of the system. The danger of modifying a software element consists of the occurring