Analyzing and modeling the structural and qualitative interdependencies of software evolution

Adeel Ahmad, Henri Basson, Mourad Mohamed Bouneffa

ISBN 978-1-84626-xxx-x
Proceedings of 2010 International Conference on XXXXXXXXXXXXXXX
(xxxxxx2010)
Singapore, *26-28 February*, *2010*, *pp. xxx-xxx*

Analyzing and modeling the structural and qualitative interdependencies of software evolution

Adeel Ahmad, Henri Basson, Mourad Bouneffa

Laboratoire d'Informatique, Signal et Image de la Côte d'Opale
BP-719 62228 CALAIS Cedex FRANCE
{ahmad, basson, bouneffa}@lisic.univ-littoral.fr

Abstract. In the context of software evolution management, we present a global modeling approach aiming at a better analysis of a change impact and its propagation in developed software. We consider two major modeling aspects under which an intended change should be considered. These are structural and qualitative dimensions of impacts of projected software change. With the proposed modeling approach, we consider the established models for tracing the qualitative change impact flow associated to the targeted software evolution. The approach is validated with the help of a prototype framework that includes the software specific information data gathering and automatic mechanisms aiming at structural and qualitative change impact analysis. Resulting detailed analysis is destined to help the decision-making in various cases of software evolution and maintenance.

Keywords: change impact analysis; software modeling; software quality; software structure; software evolution;

1. Introduction

The term *Software evolution* lacks a standard definition. It generally refers to a continuous process activated in response to changing of user requirements or assured functionalities in current operating environments of software systems. *Software change* is constituted from the various basic operations to be executed in order to reach a targeted specific evolution of software systems. Uncontrolled changes in software may often lead to complex and numerous undesired effects.

The extent of the change or the identification of software artifacts that are impacted by the change (also referred to as the ripple effect of change) can be assessed with change impact analysis [1]. It consists of a collection of techniques for determining the ripple effects of modifications in software systems. Many of these techniques are founded on empirical studies of evolution control in monolithic software systems. This included the refactoring, reverse engineering, and re-engineering to observe the code fragments that a change may affect. These techniques can help program comprehension, but at the same time they lack an empirical evidence to prove their consistence for the change impact analysis. Henceforth, they provide very limited knowledge to describe the ripple effect of applied changes. This is especially true when quality degradation issues may become alarming to be taken into account.

A significant part of the concerned literature indicates that software engineering community is interested in preserving consistency during the evolution of software models issued from different development phases [2]. The traceability links therefore exist between software models. These links are established between abstract representations of the software and the more refined ones. While on the other hand, the change impact analysis may also be focused within a particular development model. To control software evolution, under the constraints of time and resources, without the quality degradations, with continuous changes making software more complex and larger, is a real challenge.

This cannot be achieved without having an exploitable knowledge, describing exhaustively software artifacts and the different types of relationships between them. Improving the software evolution control, without quality degradation issues, requires a deep understanding of software components interdependencies. Exhaustive information regarding different software artifacts is also required to analyze qualitative aspects of incorporated changes. In this paper, we discuss an *a priori* analysis of structural and qualitative impact propagation of an intended change. This is achieved by means of formalization of the software artifacts and their various interactions. The objective is to provide a systematic approach for successful change incorporation in the developed software and thus to minimize various risks.

The paper is organized as follows: section 2 explains briefly the structural modeling approach for software evolution; in section 3, an insight to the qualitative evaluation of software systems is given; section 4 explores the structural and qualitative change impact propagation with the help of a prototype tool; in section 5, we give concluding remarks and future prospects of the ongoing work.

2. Structural analysis of change impact propagation

A software application is generally a result of transformations through a sequence of models associated to the different phases of development life cycle. Each model, has some abstraction level, provides software description from some point of view, detailing the previous level. The structure of development models, their importance and their chronological execution depends on the specific phase of applied software development. Pragmatically, it is observed that the several models of the software such as documentations, designs, and source codes etc. simultaneously exist but only source codes are effectively maintained. Subsequently, that makes other representations inconsistent and no longer valid. Using and updating these representations of software systems should be supported by some infrastructure. Their maintenance can be controlled on some extent by complying with the Model Driven Architecture framework.

A software development model can be further decomposed on levels of abstraction or granularity in pertinence to software artifacts. It is meant by software artifacts all the objects composing the software. They may be the specification documents, design documents, database schema, or non-functional ones like test data, quality documents etc. Considering the source code model of an object-oriented language, the software artifacts can be classified on three broader levels. They can be *coarse-grained* like *applications*, *files*, or *libraries*. They can be of *medium granularity* such as *generic classes*, *attributes*, and *class method members* of *objects*. They can also be *fine-grained* such as *statements*, *expressions*, or *control structures*.

We principally identify three categories of relationship types, which serve as the basic entities responsible to provide a path for change impact flow. In *Structural Model for Software Evolution (SMSE),* we consider the vertical traceability links as the relationship type in *inter-model relationship*, while the horizontal traceability links are further decomposed either as *inter-level relationship*, or *intra-level relationship*. An *Inter-model relationship* mapping can help the consistency among different development phases. It is a bi-directional relationship type, which represents traceability links between artifacts issued from two different development phases. As discussed earlier, *SMSE* decomposes the software development phase as viewed on several levels based on the abstraction and granularity levels of software artifacts. The categorization of artifacts on different levels is refined in respect of language semantics. Therefore, in case of a change, we analyze the *inter-level* and *intra-level relationships*, within and among these *levels*, and that progressively in each model as well as between different phases [3].

The potential of a relationship type occurrence to propagate a change impact from one of the participating artifacts to another linked artifact qualifies its *change impact conductivity*. This highly depends on the relationship type occurrence and its cardinalities. The relationship cardinality encapsulates the properties characterizing the conditions and iterations of occurrences of such relationship. Hence, change impact analysis and conductivity of a relationship wouldn't be effective until all the relevant cardinalities of the relationship type are considered. In addition, cardinalities regarding the occurrence of a relationship type, can be used to identify affected artifacts and provide a set of attributes which permits to qualify, more precisely, the *change impact conductivity*.

3. Critical causal flows for the qualitative assessments

Changes in software structure may affect some previous good levels of software quality characteristics. The *Software Quality Assurance (SQA)* has always been a complicated task and deriving formulations to evaluate the whole software quality is more complicated without traceability information. The software quality engineering community has been studying quality evaluation issues for a significant amount of time and made much progress in quality metrics. There have been proposed many theoretical and numerical frameworks to assess the software quality. The software quality models proposed by B. Bohem and Mc. Call [4] are known to be as the historical models to represent, conceive, and evaluate the software quality. They have investigated the involvement of multiple quality factors to assess the whole software quality. Several variations have been formulated to the Boehm-McCall Model since its proposition [5]. These address the analysis of software quality factors and their qualitative or quantitative metrification.

The investigation of underlying forces that cause quality degradations discovers that an exhaustive description of different aspects of change impact propagation may still need improvements for better control of software evolution. It is more complicated when it comes to generate and maintain enough empirical data to qualify the interdependencies of software artifacts at specific points for tracing causal flows. Although, a need also exists to manage the information regarding storage and analysis of quality factors and inter-dependence of quality attributes during a change incorporation process in different models at different phases of software development life cycle [6].

In qualitative modeling, the *Factors* and the *Criteria* represent the quality attributes of artifacts developed at each phase within an individual model of software development. The *Metrics* are destined to their evaluations as terminal nodes in a specific devoted sub-graph per model.

The Qualitative Model for Software Evolution (QMSE) considers the quality graph of software built on several artifact levels in respective software phases. The root of the model, denoted as $QG(\Sigma\Phi.Art)$, represents the general quality of the software artifacts. We denote the individual quality of the model i at phase j as $Q_i(\Phi_j)$ and the total number of phases of the development model as $|\Sigma\Phi|$. The node ($QG(\Sigma\Phi.Art)$) is then defined such as the descendant set (Dsc):

$$Dsc(QG(\Sigma\Phi.Art)) = \{Q_i(\Phi_1.Art), Q_i(\Phi_2.Art), \dots , Q_i(\Phi|\Sigma\Phi|.Art\} \quad (1)$$

In the same way, descendants contain the roots of the sub-graphs of the quality of software artifacts from each phase. We adopt the depth varying levels of quality characteristics, breaking-down where the numbers of refinements levels (Quality Factors into Criteria, Quality Criterion into sub-Criteria, etc.) vary according to the complexity of the concerned factor or criteria (Fig.1). It may be useful to underline that there is no conceptual difference between criteria and descending sub-criteria. Both represent the technical attributes linked by a composition relationship, implying that all the sub-criteria help to directly evaluate their direct (or indirect) ancestors. The refinement process of a criterion into sub-criteria, and then recursively their sub-criteria into more detailed quality characteristic, is a task realized by the quality assurance engineer(s) of the current application. The quality engineer drives this process recursively until she specifies the precise attributes that should be considered regarding their relevance and the potential of their evaluation using one or more metrics calculation and interpretation. In the same way, the quality engineers propose the metrics destined to the evaluation of quality characteristic concerning two or several phases using a specific sub-graph recursively until the assessment of its global quality level regarding the whole software system. We also intend to define the related change impact propagations between quality factors, criteria and the sub-criteria of the same level of refinement. The representation of these horizontal relationships allows to consider the contradictions which may exist between different quality attributes. It then permits to define an evolutionary quality evaluation. Practically, in all software applications the related levels of factors and criteria are identical, these describe the general properties, contributing each, in the certain measure to the assurance of the global quality. This refinement of criteria into further sub-criteria allows to pass from a generality shared by a family of languages, formalisms, methods to the particularity of each individual artifact expressed from its specific use in the application. The same refinement reflects the knowledge of the quality expert to define the relations between criteria and sub-criteria. It also reflects the desired importance of each factor, as a function of comparative priorities of other factors. This expertise is therefore in relevance to design “how to detail the quality” by which it

shows the general description of qualitative objectives to the expression of sub-objectives in condition to their accomplishments. The number of modifications of an artifact can be an important indicator of artifact fault rate, stability, and subsequently artifact complexity. While the number of times a software artifact is executed and inter-linked with other software artifacts may indicate its importance. SMSE serves as a prerequisite for the qualitative assessments regarding the change impact traceability and the software comprehension. We further adopt QMSE to better analyze the causal flow links in software quality attributes. It particularly refers to the traceability links in qualitative assessment of software as a result of any structural change.

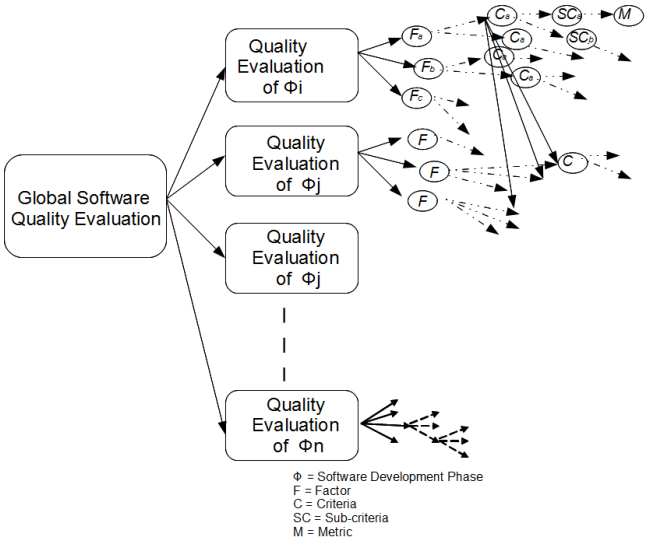

Fig. 1. Quality evaluation of software artifacts

4. Qualitative evaluation through structural changes

A structural change (ΔS) in an artifact can propagate a qualitative impact (ΔQ) on linked artifacts. ΔQ is estimated using of the variations of a set metrics in their values before and after the change ΔS. The quality engineer of evolving application adopts the metrics set to estimate its global quality. We address this particular aspect through the incremental re-formalizations to evaluate the global impact of a structural or qualitative change. For a qualitative aspect, it is important to determine the extent to which the quality of an artifact is affected by a change operated on another linked artifact. The creation of rules for change impact propagation in qualitative aspects is based on the descriptive knowledge of the cardinalities of association type among quality of software artifacts, its preliminary evaluation of conductivity, and change in metric values. More precisely, suppose that given an artifact C_x, there exists an occurrence of structural and qualitative association. We can derive a general rule concluding on the existence of change impact conductivity and flow of its propagation in the qualitative aspect. Let us denote $\Delta S \Rightarrow \Delta Q$ as the notation, means that the structural change ΔS implies a qualitative change ΔQ. The rule formation follows the *ECA (Event-Condition-Action)* principle for the inference and assertion of facts. The qualitative dependency implies the primary rule as:

If <Structure of C_x is changed> then <Quality of C_x is concerned> endif

We further elaborate the concept with the help of an example. To illustrate the impact of a structural change to qualitative model, let us consider the code snip shown in Fig. 2. For the purpose of explaining the qualitative impact propagation, we measured and observed the performance of `findEdge` (method) of `Graph_Demo` (class). We can determine the qualitative change ΔQ affected by structurally changed artifact edges of type `Collection` (data type). We invoked a change on edges for its initialization from an `ArrayList` (data structure) to `Vector` (data structure). We observed the performance measure of `findEdge` by executing it repeatedly before and after the structural change. We found a difference of performance during executions of the method (qualitative impact) before and after the change. As discussed earlier, any structural change (ΔS) invoked on an artifact may cause a relative qualitative change (ΔQ) in the corresponding quality criteria. The impact of ΔQ propagated progressively to the precedent criteria and then to factors can be traced up to the global quality of the software. We applied the qualitative performance metric on the other related `calling methods` before and after the same structural change. We found a corresponding qualitative impact propagation

of performance from this structural change, as a difference of execution time (quality metric) whereas there wasn't any structural change impact propagation. The change impact propagation of relative quality association can be then individually analyzed.

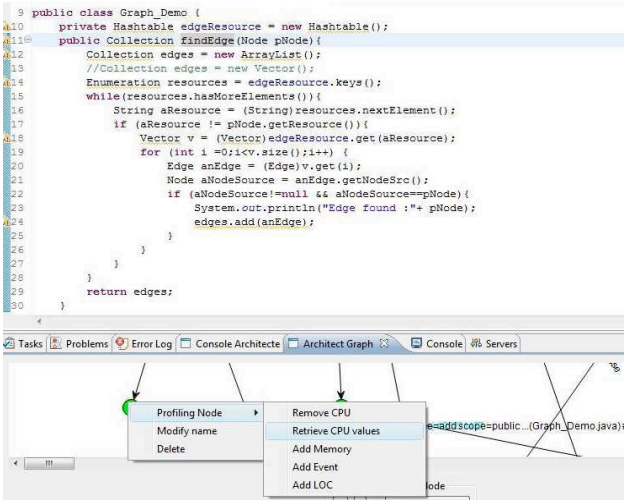

Fig. 2. Tracing the qualitative impact propagation through structural change

5. Conclusion

Software evolution is a continuous process with the incorporation of several changes. Any change in a part of software system may propagate its impacts to the other parts. It is therefore essential to control the propagation of change impact with its successful analysis. It often requires to understand the change and its impact propagation in different aspects. Further more, it is inevitable to qualify the relationship dependencies among the different elements of software from structural and qualitative point of views. A structural change in software can be analyzed on multiple levels of abstraction and granularity of software and in its qualitative impacts. We evaluate the qualitative change impact propagation among different quality factors, criteria, and the refined sub-criteria until the metrical measures, to qualify the qualitative dependencies. In this context, we present a modeling approach aiming at understanding the change impact propagation in structural and qualitative aspects. We define an overlaying meta-model for an *a priori* change impact analysis, which can help the software experts to better control the changes even before their incorporation. In the future, we intend to continue this work using the feedbacks of software evolution engineers about some selective analysis of local change impact and its propagation along with their effects on the global structural and qualitative evolution of software.

6. References

[1] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou and P. Avgeriou, "Introducing a Ripple Effect Measure: A Theoretical and Empirical Validation," *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Beijing, 2015.

[2] Shiva Nejati, Mehrdad Sabetzadeh, Chetan Arora, Lionel C. Briand, and Felix Mandoux. 2016. Automated change impact analysis between SysML models of requirements and design. In *Proceedings of the 24th ACM SIGSOFT International Symposium* FSE 2016. ACM, New York, NY, USA, 242-253.

[3] A. Ahmad, H. Basson, L. Deruelle, and M. Bouneffa, "Towards a better control of change impact propagation," in *INMIC'08: 12th IEEE International Multitopic Conference*. IEEE Computer Society, December 2008, pp. 398–404.

[4] J.A.Mccall, J.P.Cavano, and G.Walters,"Factors in software quality," vol. 1, 2, and 3, November 1977.

[5] Lewis, William E. Software testing and continuous quality improvement. CRC press, 2016.

[6] A. Ahmad, H. Basson, L. Deruelle and M. Bouneffa, "Towards an integrated quality-oriented modeling approach for software evolution control," *2010 2nd ICSTE*, San Juan, PR, 2010, pp. V2-320-V2-324. doi: 10.1109/ICSTE.2010.5608798

Authors' background

Your Name	Title*	Research Field	Personal website
Henri Basson	Professor	Software Engineering	http://www-lisic.univ-littoral.fr/spip.php?article50&membre=4
Adeel Ahmad	Associate Professor	Software Engineering	http://www-lisic.univ-littoral.fr/spip.php?article50&membre=1
Mourad Bouneffa	Associate Professor	Software Engineering	http://www-lisic.univ-littoral.fr/spip.php?article50&membre=9