ijecs open access www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 4 Issue 4 April 2015, Page No. 11126-11128

Category, Strategy and Validation of Software Change Impact Analysis

Zeba Mahmood* Rakesh Bharti Tahera Mahmood
CSE&UIT Allahabad,India
mahmood.zeba@gmail.com
goswami.rakesh@gmail.com
mahmood.tahera@gmail.com

Abstract- *Impact analysis is defined as the process of identifying the potential consequences (side-effects) of a change, and estimating what needs to be modified to accomplish a change. We propose a UML model-based approach to impact analysis that can be applied before any implementation of the changes, thus allowing an early decision-making and change planning process. We first verify that the UML diagrams are consistent (consistency check). Then changes between two different versions of a UML model are identified according to a change taxonomy, and model elements that are directly or indirectly impacted by those changes (i.e., may undergo changes) are determined using formally defined impact analysis rules (written with Object Constraint Language). A measure of distance between a changed element and potentially impacted elements is also proposed to prioritize the results of impact analysis according to their likelihood of occurrence.*

Keywords— Software Change Impact Analysis (SCIA), impact analysis, UML diagrams.

INTRODUCTION

Software change occurs for several reasons, for example, in order to fix faults, to add new features or to restructure the software to accommodate future changes. Changing requirements is one of the most important motivations for software change [1]. Requirements change from the document design to the testing phase. Changes to requirements reflect how the system must change in order to be useful for its users and provide better service from other products available in the market. At the same time, such changes impose a great risk as they may cause serious hazard especially in category of embedded software and related to defense and medical fields [2]. Thus, changes to requirements must be properly managed and controlled to ensure the survival of the system from a technical point of view [3]. An organization that develops software requires a proper change management process in order to mitigate the risks of constantly changing requirements and their impact on the system. Bohner and Arnold [4] define change impact analysis as the process of identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change. Several authors address change impact analysis in the context of requirements modeling. Common techniques used to implement change impact analysis are based on either traceability or dependency relationships between the software artifacts. Traceability-based impact analysis techniques work on analyzing the relationships between requirements and other development artifacts (such as design, implementation and test cases) to determine the scope of the anticipated changes, dependency-based impact analysis techniques work on a more detailed level by analyzing the relationships between the artifacts of the same development phase. Lehnart et al. [5] proposes impact analysis technique's classification

A. *Code Based Software Change Impact Analysis*

Code based approaches investigate the impacts of changes by reasoning about inheritance relations, method call behavior, and other dependencies between source code entities. Source code programs, class packages, classes, methods, statements, and variables are analyzed to predict the propagation of changes. However, such techniques are not applicable in the early stage of software design and requirements analysis, when no source code is available. Their source code dependent nature also limits their application to programmers and project leaders. Static code analysis extracts facts from source code to build call graphs, slices, and other representations which are used to assess the impacts of
a change.

B. *Model Based Software Change Impact Analysis*

Model based analysis provides facility of SCIA for software models to keep their quality and correctness at early stage. Models, such as UML diagrams, enable the assessment of architectural changes on a more abstract level than source code. This enables SCIA in earlier stages of development and in MDD, which has become more important in recent years. But dependent on the underlying modeling language, even model based analysis provides effective impact analysis results, for example when analyzing detailed UML class diagrams.

C. *Artifacts Based Software Change Impact Analysis*

Miscellaneous artifact based SCIA is combined form of source code and model based techniques including some other software artifacts. Different researchers have used different artifacts for impact analysis e.g. some using code and SRS both and some UML models and source code both.

CONTRIBUTING FACTORS FOR CHANGE REQUEST

Factors that can inflict changes to requirements during both initial developments as well as in software evolution are

1. The objective of the system is supposed to frequently change, for example for economic or technological reasons.
2. The users change their requirements about what they want the system to do, as they understand their needs better later than the beginning. This can happen because the users initially were uncertain about what they want.
3. The specifications of the system changes. For example, increases in processor speed and capacity of computers can affect the expectations of the system.
4. The new system is developed and released for beta testing to leading users to discover new requirements.

STEPS TO VALIDATE CHANGE REQUEST

The support of impact analysis of UML design models can be decomposed into several sub-problems:

A. Verify the consistency of changed diagrams

The original and modified models must be self-consistent for any impact analysis algorithm to provide correct results. Note that this is different from impact analysis as it does not focus on finding (potentially) impacted elements (i.e., whose implementation may require change) between two model versions, but structural inconsistencies between UML diagrams of a single model, e.g., a class instance (classifier role2) in a sequence diagram whose class is not in the class diagram. Since consistency in complex UML models is not always easy to achieve, verifying consistency must be supported by tools. Inconsistencies may be automatically modeled and detected by a set of consistency rules.

B. Automatically detect and classify changes

Automatically detect and classify changes across different versions of UML models. Ideally, one modifies a UML model and then uses the impact analysis tool to automatically identify all the changes performed since the last version. We do not want software engineers to have to specify each and every change as we want to avoid the overhead that would prevent the practice of impact analysis. As seen below, changes have to be classified to be able to perform a precise impact analysis.

C. *Perform an impact analysis*

Perform an impact analysis to determine the *potential* side effects of changes in the design. In most cases, for reasons described below, side effects cannot be identified with certainty as there is no way to ascertain whether a change is really necessary based on the UML analysis or design only. As a result, an *impacted element* is a UML model element whose properties or implementation *may* require modification as a result of changing another model element (i.e., one of its properties may change). To clarify the terminology we employ, changes to UML diagrams are the result of *logical changes* corresponding to error corrections, design improvements, or requirement changes. We refer to *changes to model elements* when a property of an element has changed from one version of a diagram to another, e.g., the visibility of an operation. A logical change usually results in a set of changes to model elements. Impact analysis can be performed for each logical change independently or for an entire, new UML model.

D. Prioritize the results of impact analysis

Prioritize the results of impact analysis according to the likelihood of occurrence of predicted impacted elements. In object-oriented designs, when considering all direct and indirect dependencies among model elements, impact analysis often results in a large number of (potentially) impacted model elements, thus making their verification impractical. Addressing this issue requires a way to order side effects according to criteria that can be easily evaluated and which are good indicators of the probability of a side effect, for a given change. .

STRATEGY FOR IMPACT ANALYSIS

There are various strategies for performing SCIA, these strategies are based on some parameters which is considered during their requirements engineering process the execution phase. These strategies can be broadly classified as following:

1. Automatable

2. Manual

With automatable strategies, means strategies, which are based on some tools. These strategies have the ability to provide very fine-grained impact estimation in an automated manner, but these strategies require detailed infrastructure and related information of projects [20]. With manual strategies, means those that are best performed by human beings (not by tools). These strategies require fewer infrastructures, but may be harder in their impact estimation than the automatable ones.

CONCLUSION

In this paper we are discuss Software Change Impact analysis and their category.1) Code based approaches investigate the impacts of changes by reasoning about inheritance relations, method call behavior, and other dependencies between source code entities.2) Model based analysis provides facility of SCIA for software models to keep their quality and correctness at early stage . Also discuss Strategy for SCIA that is automated and manual. And discuss validation of Impact Analysis.

REFERENCES

[1] R. S. Pressman. Software Engineering: A Practitioner's Approach. McGraw-Hill Higher Education, 5th edition, 2001

[2] P. Bengtsson and J. Bosch. Architecture level prediction of software maintenance. In Software Maintenance and Reengineering, 1999. Proceedings of the Third European Conference on, pages 139–147. IEEE, 1999.

[3 A. G¨oknil, I. Kurtev, and K. van den Berg. Change impact analysis based on formalization of trace relations for requirements. 2008.

[4 R. Arnold and S. Bohner. Impact analysis-towards a framework for comparison. In Software Maintenance ,1993. CSM-93, Proceedings., Conference on, pages 292– 301, 1993

[5] S. Lehnert. A taxonomy for software change impact analysis. In Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution, IWPSE-EVOL '11, pages 41–50, New York, NY, USA, 2011. ACM

[6] A. Marcus, A. Sergeyev,V. Rajlich,and J. I. Maletic. An information retrieval approach to concept location in source code. In *Reverse Engineering, 2004. Proceedings. 11th Working Conference on,* pages 214–223. IEEE, 2004

[7] Y. Li, J. Li, Y. Yang, and M. Li. Requirement-centric traceability for change impact analysis: A case study. In Q.Wang, D. Pfahl, and D. Raffo, editors, *Making Globally Distributed Software Development a Success Story, volume 5007 of Lecture Notes in Compute Science,* pages 100–111. Springer Berlin Heidelberg, 2008.