CQV-UML Tool: a tool for managing the impact of change on UML models

Dhikra Kchaou
Mir@cl Laboratory,
University Of Sfax, Sfax, Tunisia
Dhikra.kchaou@fsegs.rnu.tn

Nadia Bouassida
Mir@cl Laboratory,
University Of Sfax, Sfax, Tunisia
Nadia.Bouassida@isimsf.rnu.tn

Hanêne Ben-Abdallah
King Abdulaziz University,
Jeddah, KSA
hbenabdallah@kau.edu.sa

***Abstract*— An automated change impact analysis and management approach is vital to handle the complexity of adapting software during its evolution. Such an approach reduces the maintenance cost and provides for adequate decision making when confronted with the choice of accepting or ignoring changes. This paper presents a change impact management approach between UML models. It verifies the consistency and the quality of interdependent diagrams after a change is handled. In addition, it calculates the effort required in managing any change and displays a report indicating to the designer all necessary modifications to keep the design coherent. The approach is supported by a toolset, called CQV-UML tool.**

Keywords—change impact; consistency; quality; effort estimation

I. INTRODUCTION

With the continuous evolution of software systems, it comes the need to automate the process of analysing and managing the change impact both on the necessary model updates and quality. Change impact analysis and management of interdependent models while keeping their quality is necessary.

A change impact analysis (CIA) method must firstly identify changes made by the designer. Such precise definition provides for the identification of the change operations and the elements affected directly by this change. Secondly, a CIA method must provide for the needed traceability between the different elements in order to analyse the impact of a change not only in the changed model but also between the interdependent models. Based on the identified change and the traceability between elements, the violated consistencies must be detected using a set of consistency rules. The objective of a change impact management approach should cover not only the detection of inconsistencies, but also the ability to provide for a means to estimate both the effort required to handle a change and its impacts on the quality of the various models. A tool that automates this process while taking into account the quality of the changed models and the effort needed represents a success factor of a change management method.

Several change impact management methods for UML diagrams have been implemented (e.g., [1], [3], [5]). They tried to assist the designer in correcting inconsistencies caused by change. However, these techniques do not evaluate the quality of UML diagrams after evolution. In addition, the effort estimation is not treated.

In this paper, we present the tool support CQV-UML tool to automate the impact of changes on UML models. The tool aims to produce a report with warnings about every change that may deteriorate the quality of the model, recommendations about how to make the design consistent, and the number of corrections required to ensure the consistency of all the models. The designer can use this report to decide about which changes should be rethought and/or canceled. Moreover, assisted by a set of quality rules based on metrics, the designer would have an important support in producing a good quality design, which is an essential determinant of the success of the software project.

The remainder of the paper is organized as follows. Section 2 presents an overview of the existing change impact analysis tools. Section 3 presents our change impact analysis and management approach. Section 4 illustrates the usage of the tool through the T-Rot example. Section 5 summarizes the paper and outlines future work.

II. EXISTING UML CHANGE IMPACT ANALYSIS TOOLS

An idea that all researchers involved in change impact analysis agree on is that a manual change impact analysis is too expensive and error prone and that tool support is necessary. In this section, we briefly present change impact analysis tools that have been proposed in the literature. Essentially, we are interested in tools for change impact management applied to UML models.

For instance, Briand et al., [1] propose the iACMTool which manage the change between class, sequence and state chart diagrams by identifying specific change propagation rules for all types of changes. In order to assist the designer to decide about the change, they propose a measure of distance between a changed element and potentially impacted elements to prioritize the results of impact analysis. However, due to the large number of UML model element types and the large number of change types, the number of impact analysis rules is quite large which makes the process of change impact not easy to implement.

Egyed [3] extends the tool in order to assist the designer in discovering unintentional side effects, locating choices for fixing inconsistencies, and then in changing the design model. In fact, for a given inconsistency and for each element in the scope elements, the tool enumerates a list of choices to correct the change. Since there is a large number of choices, the author tries to reduce this list based on the reason of the inconsistency. However, the number of choices as well as the choices

DOI reference number: 10.18293/SEKE2015-040

themselves may be ambiguous for the designer especially with for large diagrams.

Keller at al., [5] present an inconsistency resolution framework implemented as an Eclipse plug-in. The tool takes as input one type of change applied to one or more UML model element and outputs a set of impacted elements. In fact, for a given changed element, the tool implements seven impact analysis rules suitable for seven change types. The tool does not give any support to help the designer to correct the change.

Overall, existing works try to detect inconsistencies caused by changes in order to ensure the consistency within and inter UML diagrams. However, they do not offer any assistance to correct detected inconsistencies and to preserve the quality after the change. In addition, they do not support the generation of the corrected diagrams.

III. THE CHANGE IMPACT ANALYSIS AND MANAGEMENT APPROACH

Our approach allows the identification and measurement of potential side effects resulting from requirement/design changes. Its first originality is that it provides traceability between documented use case, class and sequence diagrams. The second originality of our approach is that, besides verifying the consistency of changed UML diagrams, it verifies their quality using a set of metric based quality rules. The third originality is that it measures the effort needed to manage inconsistencies in order to assist the designer in deciding about which changes should be rethought and/or canceled. The fourth originality is that it is supported by a tool that automates all steps and that automatically generates a new version of the corrected diagrams after resolving the inconsistencies caused by different changes.

A change impact analysis technique needs first of all a way to specify changes. Thus, in a previous work [4], we defined a MOF [2] based change meta-model that covers all change types at a high level of abstraction. Being MOF-based, our change meta-model defines all possible changes affecting the elements independently of a particular modeling language. In addition, it can be extended and adapted to define changes that affect any MOF-based model and, in particular, UML models.

The second hurdle that change impact analysis and management faces is the semantic and structural traceability among the numerous elements of the different diagrams. For this purpose, we propose a graph concept, called "model dependency graph" that encodes the requirements (use case) and design (class, sequence) diagrams in an integrated way. The encoding uses the semantic traceability results and explicitly represents the syntactic relationships among the diagrams' elements. The model dependency graph provides for the needed traceability to analyze systematically the impact of a change on the consistency of the diagrams.

Inspired from the work of Lallchandani et al., [6] for static slicing of UML models, the model dependency graph (MDG) is constructed by transforming the use case, class and sequence diagrams to graphs. In particular, our adaptation accounts for the association relationships (not treated by Lallchandani et al., [6]). In addition, we integrated the documented use case diagram to the MDG which is also not treated in by Lallchandani et al., [6].

In our approach, the UML class diagram is transformed into a Class Dependency Graph (CDG) and every UML sequence diagram is transformed into a Sequence Dependency Graph (SDG). Every UML use case diagram is transformed into a Use Case Dependency Graph (UCDG) based on a structured use case description [7]. To get all dependencies among the various diagrams, the UCDG, CDGs and SDGs are merged into a Model Dependency Graph (MDG).

To trace the change impact from the use case diagram across the class and sequence diagrams, the CDG and SDGs are firstly integrated into a single graph. The constructed MDG must be completed with the requirements diagram, i.e., the documented use case diagram. For this, we need to identify the correspondence among the ordered actions (specified in the use case scenarios) and the messages in the sequence diagrams. For this purpose, we use an information retrieval technique: term frequency – inverse document frequency (TF-Idf) [8] to measure the cosine similarity measure [12] in order to determine the most resembling message in the sequence diagram to the query (i.e., action in the scenario) and consequently to a use case. In our case, documents and queries contain the set of grammatical units that compose a message/action in SD/UC added to their synonyms extracted from WordNet. The calculus of the different weights for the terms is followed by the calculation of a similarity measure which is the cosine.

After the cosine similarity calculation, the documents (i.e. the actions in the UCs) that are similar to a query (i.e. messages in SD) are linked together in order to construct the MDG. Note that after this step, a validation step may be needed by the designer since the results of the cosine similarity computation may return several ranked possibilities. The designer should validate/select one value that better fits his situation.

Once the change is detected and the traceability is established, the consistency of changed diagrams is verified based on a set of consistency rules. As an example of an inter-diagram consistency rule, each operation in SD must be defined in the receiver's class in CD. A change may affect not only the consistency of UML diagrams, but also their quality. Thus, in addition to the preservation of the consistency of UML diagrams, their quality must be evaluated and preserved after a change is introduced. For instance, the deletion of a class that had many important relationships with other classes, or that participates in a design pattern [10] depreciates the quality of the class diagram. The quality of software can be evaluated using several metrics (cf., [9]) interpreted through a set of thresholds (cf., [11]). In our approach, these metrics and their thresholds are used to evaluate and/or to predict the quality effects of a change in order to propose a set of recommendations to the designer. For this objective, we propose a set of quality rules. To preserve the intra-diagram quality after a change, we calculate the CK metrics suite [9] before and after every change, and based on their thresholds [11], we verify a set of quality rules and we inform the designer about any violation. On the other hand, in order to approximate the effort needed for change impact management, we propose to calculate the number of

required modifications (NRM) necessary when correcting the inconsistencies caused by an intra/inter diagram changes. The NRM sums up the number of update/change operations needed to correct a violated consistency rule.

To calculate NRM, we propose the new intra/inter diagram metrics shown in Table 1. Note that the proposed thresholds for these metrics will be defined using an empirical study in a future work. For a given change, the NRM is the sum of the metrics values, concerned by the change.

TABLE 1: METRICS USED TO MEASURE THE INTRA/INTER-DIAGRAM QUALITY

Metrics		Definitions
Intra-diagram metrics	**NAt/Op**	Number of times an attribute (and operation) of a class is used (called) in an operation.
	NAtPr	Number of times an attribute is used as a parameter in an operation.
	NM2Ob	Number of messages between two objects.
Inter-diagram metrics	**NCSD**	Number of times a class is used as an object in SDs.
	NATSD	Number of times an attribute is used in SDs.
	NOpSD	Number of times an operation is used as a message in SDs.

IV. CQV-UML TOOL: A TOOL SUPPORT FOR CHANGE MANAGEMENT

To implement our change impact management approach, we have developed a tool named CQV-UML Tool: a Consistency and Quality Verification tool for UML diagrams.

A. Functional architecture

The principal activities performed by our tool are the change detection, the consistency verification and quality verification. The tool takes as input a set of UML models versions corresponding to the original and changed diagrams modelled using the CASE Tool: ARGOUML. The first step transforms the XMI files corresponding to the design diagrams into XML. The transformation is performed thanks to XSLT to obtain reduced representations using the API JDOM. The aim of this step is to eliminate all superfluous information, that is specific to the CASE tool ARGOUML. Secondly, the list of changes is recorded and displayed to the designer. Afterwards, the cosine similarity measure is calculated and our graph based technique (MDG) is implemented in order to establish the traceability between the interdependent diagrams. Based on the achieved traceability, the set of violated consistency and quality rules corresponding to each change type is displayed to the designer. Finally, after accepting the proposed correction, the corrected diagram is generated.

B. CQV-UML tool application

To illustrate the various functionalities of the CQV-UML Tool including the consistency and quality verification of the different diagrams after evolution, let us consider a case study where UML was used to develop a system for autonomous navigation by the intelligent service robot, T-Rot. The use case diagram comprises two use cases (Figure 1): "Navigation" and "Obstacle Avoidance". The "Navigation" use case textual description is presented in Table 2. The sequence diagram SD corresponding to the Navigation use case is presented in Figure 2. The class diagram CD of this example is presented in Figure 3.

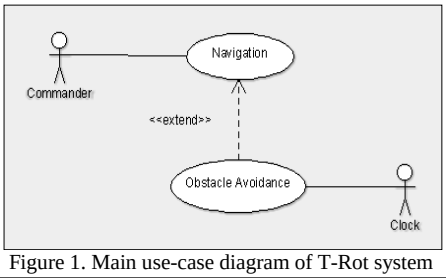

Figure 1. Main use-case diagram of T-Rot system

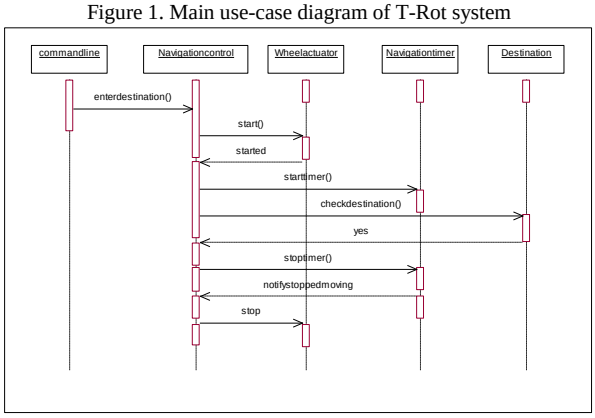

Figure 2. SD1: the sequence diagram of the UC1 "Navigation"

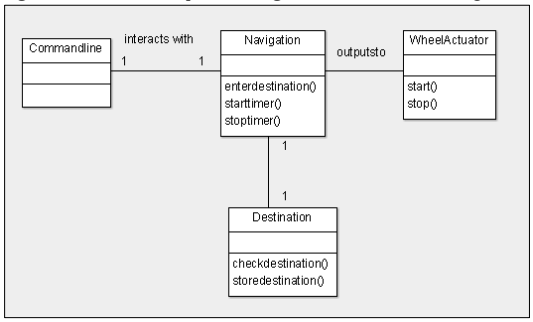

Figure 3. The Robot system class diagram CD

The first step in our approach consists in detecting changes for each UML diagram. The interface 1 of Figure 4 shows the detected changes in the class diagram. By clicking on the change (delete the *Start()* operation from the *WheelActuator* class in the CD), the list of violated consistency rules in the top-right of the screen shot are displayed. In fact, the change violates the consistency rule: Each operation in SD must be defined in the receiver's class in CD since the deleted *start()* operation exists in the SD1 between the *Navigationcontrol* object and the *WheelActuator* object. This inconsistency is detected thanks to the traceability obtained through the MDG. The CQV-UML tool recommends the designer to correct this inconsistency by deleting the message *start()* in SD1 corresponding to the deleted operation or undo the change. To take the appropriate decision, the CQV-UML tool calculates the number of required modification NRM needed to correct the inconsistency. The NRM is calculated based on the metric NOpSD: number of times the operation is used as a message in SDs. The *start()* operation exist one time in the SD1, so, the designer has one delete to do.

To manage the second change, the CQV-UML tool calculates the similarity measure between the deleted action

NSa3 in the *Navigation* use case and the list of messages in the sequence diagram corresponding to the *Navigation* use case. The interface 3 of Figure 4 presents the similarity results which show that the *NSa3* action corresponds to the *notifytostopmoving* message in SD1. The system informs the designer that the deleted action exists as a message in SD and as an operation in CD and proposes to delete them.

TABLE2. "NAVIGATION" USE CASE DESCRIPTION (UC1)

Use case	Navigation
Actor	Commander
Precondition	The robot system has the grid map and the current position.
PostCondition	The robot system is at the destination and waiting for the next destination
Extension Point	[obstacles are recognized], use case « Obstacle Avoidance »
Normal Scenario NS	<NSa1><The user > <enters a destination > <NSa2><The system> < commands the wheel actuator to start moving to the destination > < NSa3>< The wheel actuator > < notifies the system that it has started moving > < NSa4>< The system> <determines that it arrives at the destination> < NSa5>< The wheel actuator><notifies the system that it has stopped moving >
Alternatives Scenario AS	< If the system doesn't arrive at the destination > <AS1a1> <the system> < keeps moving >

The quality verification in the interface 2 of Figure 4 shows that the deletion *outputsto* association from CD violates a quality rule. In fact, the *WheelActuator* class becomes isolated. The tool recommends to the designer to add an association or to cancel the change.

When the designer accepts the proposed corrections, the affected diagrams are modified and displayed to the designer. The interface 4 of Figure 4 shows the generation of the corrected class diagram.

V. CONCLUSION

This paper introduced an approach for change impact management and its associated CQV-UML tool. The proposed approach consists in managing the impact of changes affecting elements in UML diagrams essentially use case, class and sequence diagrams. The MDG graph is used to model the inter-dependencies between the different diagram elements and as a consequence to trace the impact of the change.

We are currently examining how to evaluate the proposed approach on open-source systems and comparing the results of our experiments with other existing approaches.

REFERENCES

[1] L. C. Briand, Labiche, Y. O'Sullivan, L. Impact Analysis and Change Management of UML Models‖. In Proceedings of the International Conference on Software Maintenance, 2003, pp. 276-280.

[2] OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1, OMG Document Number: formal/2011-08-07, http://www.omg.org/spec/MOF/2.4.1/PDF.

[3] A. Egyed, Fixing Inconsistencies in UML Design Models. Proceedings of the 29th International Conference on Software Engineering, 2007, pp. 292-301.

[4] D. Kchaou, N. Bouassida, H. Ben-Abdallah, A MOF-based change meta-model", Proceedings of the International Arab Conference on Information Technology, CCIS, Zarqa, Jordon, 2012.

[5] A. Keller, S. Demeyer, Change Impact Analysis for UML Model Maintenance‖. Book chapter: Emerging Technologies for the Evolution and Maintenance of Software Models, 2012, pp. 32-56.

[6] J.T. Lallchandani, R. Mall, Static Slicing of UML Architectural Models, Journal of object technology, Vol. 8, No. 1, 2009, pp. 159-188.

[7] M. Ali, H. Ben-Abdallah, F. Gargouri, Towards a Validation Approach of UP Conceptual Models, In : Proceeding of Consistency in Model Driven Engineering in European Conference on Model Driven Architecture - Foundations and Applications Nuremberg, Germany, 2005, pp. 143-154.

[8] H. Wu and R. Luk and K. Wong and K. Kwok. "Interpreting TF-IDF term weights as making relevance decisions". ACM Transactions on Information Systems, 26 (3). 2008.

[9] S.R. Chidamber, C.F. Kemerer, Towards a metrics suite for object oriented design. In Conference proceedings of Object-oriented programming systems, languages, and applications, 1991, pp. 197-211.

[10] Bouassida N., Ben-Abdallah, Issaoui I. Evaluation of an automated multi-phase approach for pattern discovery ", International Journal of Software Engineering and Knowledge Engineering, World Scientic, Vol 23, N10, pp 1367-1398 (2013).

[11] E. Chandra, P. Linda, Class Break Point Determination Using CK Metrics Thresholds‖, Global Journal of Computer Science and Technology, Vol. 10, 14, 2010, pp. 73-77.

[12] A. Singhal, Modern Information Retrieval: A Brief Overview. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24, 2013, pp. 35–43.

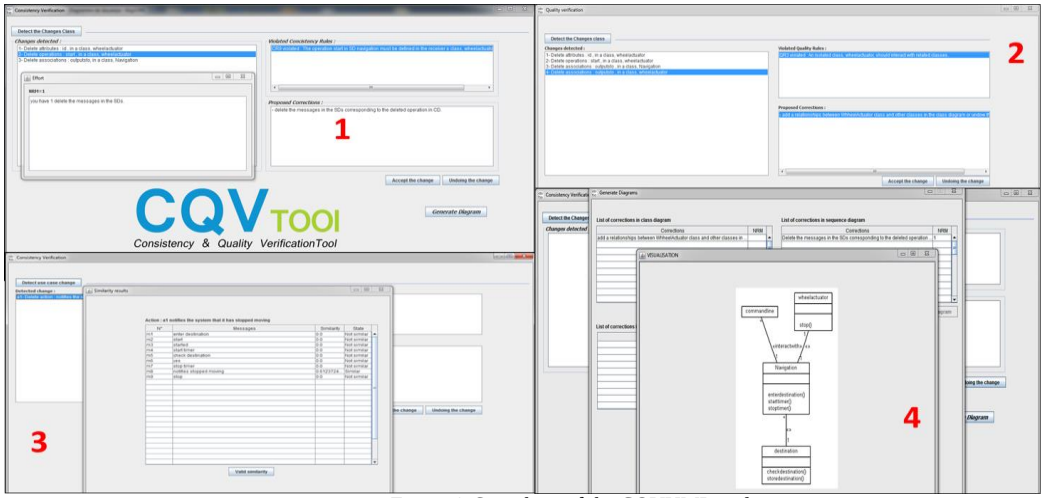

Figure 4: Snapshots of the CQVUML tool