Current Issues in UML Diagrams Coevolution and Consistency Techniques and Approaches

Bassam Atieh Rajabi*, Sai Peck Lee
Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia.

* Corresponding author. Email: bassamrajabi@gmail.com
Manuscript submitted September 03, 2017; accepted November 24, 2017.
doi: 10.17706/ijcee.2018.10.2.158-173

Abstract: UML modelling language is widely adopted in software analysis and design. UML diagrams are divided into different perspectives on modelling a problem domain. Preserving the coevolution among these diagrams is very crucial so that they can be updated continuously to reflect software changes. Decades of research efforts have produced a wide spectrum of approaches in checking the coevolution among UML diagrams. These approaches can be classified into direct, transformational, formal semantics, or knowledge representation approaches. Formal methods such as Coloured Petri Nets (CPN) are widely used in detecting and handling the coevolution between artifacts. Although ample progress has been made, there still remains much work to be done in further improving the effectiveness and the accuracy of the state-of-the-art coevolution techniques in managing changes in UML diagrams. In this research, a survey about the approaches in maintaining the coevolution among UML diagrams is provided. This research proposed set of coevolution and change effect relations on UML diagrams and diagrams elements. Additionally, currently challenges and issues to detect and resolve the UML diagrams coevolution and inconsistencies are discussed.

Keywords: Coevolution, change impact, consistency, UML.

1. Introduction

Software change is continuous and unavoidable due to rapidly changing requirements across software systems. Software change management describes a software system's ability to easily accommodate future changes. An effective change management will lead organisations to the path of success, and it is an essential activity in the software project life cycle to keep track of changes and to ensure that they are implemented in the most effective way. Software engineers continue to face challenges in designing adaptive and flexible software systems that can cope with dynamic change where requirements are constantly changing [1], [2]. Unmanaged change may lead to increasing the testing and maintenance costs. One of the crucial challenges in software change management is to preserve the coevolution and consistency among software system artefacts [3]-[5]. Understanding the coevolution which represents the dependency between artefacts that frequently change together is important from the points of views of both practitioners and researchers. Coevolution involves both change impact analysis and change propagation between software artefacts or models, and hence, it is required to check if the change in one of the artefacts ultimately affects the other artefacts and may cause some unexpected changes in them, ensure that these changes are implemented in the most effective manner, and to maintain the consistency between

artefacts. For an efficient coevolution check, change impact analysis is an important step. A change impact analysis is the activity of analysing and determining the change effect. Identifying all components affected by the change is based on the traceability analysis which analyses the dependencies between and across software artefacts. Detecting and resolving the coevolution between software artefacts can be done through various techniques. Some of these techniques are analysing release histories or versions, source code, and software architecture level analysis [6]. There are different approaches proposed in the literature that use these techniques to manage changes in the software project life cycle including changes in software requirements, design models, and programming code. Many of these approaches are focused on the coevolution of software modelling, in particular, Object Oriented (OO) software modelling, due to its wide adoption in software modelling and design.

The use of OO diagrams in modelling a software system leads to a large number of interdependent diagrams. OO diagrams are divided into different categories or perspectives; each category focuses on modelling a different perspective of a problem domain. One of the critical issues in providing a change management technique for OO diagrams is to preserve the coevolution among these diagrams so that they can be updated continuously to reflect software changes [3]-[6], [7], [8]. Unified Modelling Language (UML) is the standard language for modelling OO software. The coevolution and inconsistencies between UML diagrams is high. One of the crucial issues in checking the coevolution among OO diagrams is to control the change and to keep these different views or perspectives consistent [3], [9], [10]. Spanoudakis & Zisman [11] define consistency as "a state in which two or more overlapping elements of different software models make assertions about the aspects of the system they describe which are jointly satisfiable".

Decades of research efforts have produced a wide spectrum of approaches and techniques in checking the coevolution and inconsistency among UML diagrams. These approaches can be classified into direct, transformational, formal semantics, or knowledge representation approaches. Direct approaches use the constructs of OO and Object Constraints Language (OCL) [12], [13]. Transformational approaches derive a common notation by transforming one model to another [14], [15]. Formal approaches develop formal semantics for the OO diagrams, while knowledge representation approaches use description logics as a representation language [16]. A hybrid approach is a combination between two or more different type of these approaches [1]. According to Lucas *et al.* [8], 75% of the approaches and techniques used for detecting and handling the coevolution and inconsistencies problems are formal. The most common formal methods used are state transitions methods such as Petri Nets (PNs). In this research, a survey about the approaches in maintaining the coevolution among UML diagrams is provided. This research proposed in this section set of coevolution and change effect relations on UML diagrams and diagrams elements. Additionally, comparative studies and currently challenges and issues to detect and resolve the UML diagrams coevolution and inconsistencies are provided. In this section, the research context along with the research problem and motivation are provided. The rest of the paper is organized as follows: Section 2 introduces the software change and coevolution, Sections 3 discussed the approaches related to the coevolution and consistency in UML diagrams, Section 4 is dedicated to the approaches analysis and discussion. And finally, conclusions are drawn and suggested recommendations for some potential future research areas are highlighted.

2. Software Change and Coevolution

Software change is a strategy-driven organisational initiative to improve and redesign processes to achieve competitive advantage in performance. Change occurs frequently due to the specification at design time is incomplete due to lack of knowledge; Errors, or exceptional situations can occur during execution. The nature of the change could be ad hoc, corrective, or evolutionary. Corrective changes are implemented

to correct a design error, or to react to an exception which happens during the execution. Evolutionary changes are required due to the redesign or reconfiguration of models. Ad hoc changes are a non pre-defined action. Software change management is an essential discipline for IT organisations and enterprises. The main stages for software change management are to understand the changed elements impacted, redesign, and implementation. An efficient mechanism for controlling and managing versions is required in a change management process. Understanding the coevolution which represents the dependency between artefacts frequently change together is important from the points of views of both practitioners and researchers. Coevolution is also considered a change propagation between diagrams at the same level of abstraction [17].

Maintaining the coevolution and consistency between OO design elements could help practitioners to realize their maintenance tasks. Software changes are one of the main reasons for inconsistency problems in UML diagrams, where the change in one diagram element should be reflected to other diagrams. A consistency is usually linked to the existence of multiple models or views which participate in the development process, where the set of activities for detecting and handling consistency problems are called inconsistency management [8]. Change impact and traceability analysis are very important in solving the coevolution and inconsistency problems between UML diagrams. Software change impact is defined as: "The determination of potential effects to a subject system resulting from a proposed software change" [18]. Change impact analysis identifies the scope of modifications necessary in response to a change. Traceability analysis is required to analyse of the dependencies between and across software artefacts. Dependency analysis and traceability analysis are the two primary methodologies for performing impact analysis [19]. Traceability and consistency types are discussed in [20]. Vertical traceability refers to the ability to trace dependent artefacts within a model, while horizontal traceability refers to the ability to trace artefacts between different models within the same version. Evolutionary traceability indicates the consistency between different versions of the same model. A Semantic and syntactic consistency is based on the semantic meanings and specifications defined by UML meta-model.

Change impact and traceability analysis approaches are divided in code based and model based change impact analysis [21]. Code-based impact analysis techniques require the implementation details of a change request or a precise change implementation plan prior to determining change impacts [22]. Model-based impact analysis techniques identify and determine change impacts without using program code, and make proper decisions before considering any change implementation details [22]. Model-based techniques identify change impacts by tracking the dependencies of software objects and classes within abstract models of the software design [22]. According to Lehnert [23], assessment of model changes on a more abstract level than source code, will enable impact analysis in earlier stages of development, which has become more important in recent years. Some approaches are considered combined approaches between model-based and code-based change impacts. There are many survives and studies conducted in the area of consistency management and change impact analysis techniques in UML diagrams. Some of these studies are provided in [1], [8], [17], [24]-[27]. Some approaches related to the code based and model based change impact and traceability analyses are summarized in Table 1 and Table 2.

Table 1. Summary of Model Based Impact Analysis Techniques

Approach	Approach Description
C.-Y. Chen *et al.* [22]	An approach for performing change impact analysis is presented to describe changeable items (objects, attributes, and linkage) and their relations. Tracking the dependencies of software objects and classes within abstract models of the software design.
Mohan *et al.* [28] and Park *et al.* [29]	A process slicing approach to find change impact in processes and activities is discussed. The process slicing approach is designed to formally operate on the software process considering multiple perspectives such as behavioural, informational, and organisational perspectives. Traceability check based on software artefacts relationships.

Approach	Approach Description
De Lucia *et al.* [30]	Analyse the role of traceability relations in impact analysis. Additionally, analyse the impact based on the relations between different artefacts.
In Ekanayake , *et al*. [31]	UML class and sequence diagrams are translated into XML Metadata Interchange format and then an algorithm to check the consistency among these two diagrams is applied.
Reder and Egyed [32].	The purpose of this research is to increase the performance of incremental consistency check. It focuses on the parts that are affected by model change not on how to the validate design rules.
Ali *et al.* [33]	Ensure the validity of the conceptual model (class diagram) at the design stage using Object constraints Language (OCL).
Ibrahim *et al.* [34]	Use case driven based rules in ensuring consistency of UML model consistency rules are formalized using logical approach.
Egyed[24],[25], Elaasar and Briand [35], and Millan, Sabatier, Le Thi, Bazex, Reder and Egyed [36], and Percebois [37]	Ensure the consistency between UML diagrams using OCL.
Shinkawa [7]	Consistency check between use case, activity, sequence and statechart diagram using CPNs.
Gongzheng and Guangquan [38]	Checking the consistency between state chart and sequence diagrams in UML 2.0. XYZ/E formal language [39] which is based on temporal logic [40] is used in the consistency check.
Isaac and Navon [41]	Graph-based algorithms are used to identify which elements are affected by a change.
Puissant *et al.* [3]	Artificial intelligence technique using both generated models and reverse-engineered models of varying sizes is used to resolve the inconsistencies in UML models

Table 2. Summary of Some Code Based Change Impact Analysis Techniques

Approach	Approach Description
Weiser [42]	Process slicing approach is used to find the change impact in processes and activities.
J. Zhao [43] and [44]	Program slicing technique is used to determine the change impact.
Xing and Stroulia [45]	Analysing the design evolution of OO software from the logical view using Java programming. This research focuses on detecting evolutionary phases of classes.
Gethers *et al.* [46]	Perform impact analysis from a given change request to source code.
Costanza [47], Malabarba *et al.* [48], Vandewoude and Berbers [49]	Runtime updates based on Java programming
Huang and Song [50]	Java programming is used to perform dependency analysis between OO entities.
Kagdi, Gethers, and Poshyvanyk [19]	Combines conceptual and evolutionary techniques are used to support change impact analysis in source code.
X. Sun, Li, Tao, Wen, and Zhang [51]	Analyse impact mechanisms of different change types in Java programming.
Torchiano and Ricca [52]	Source code comments and change logs in software repository are used to analyse change impacts.
Kung *et al.* [53]and Zalewski and Schupp [54]	Concerned with code changes in OO libraries

3. Coevolution and Consistence Management Approaches

In the following sub-sections, approaches from the literature about the UML diagrams coevolution, inconsistency problem, and change impact and traceability analysis will be reviewed and discussed. These approaches will be classified into direct, transformational, formal semantics, or knowledge representation approaches. Additionally, the diagramming tools that support the coevolution and consistency management

will be reviewed. Comments and discussion about these approached are provided in of each section.

3.1. Direct Approaches

Object constraints Language (OCL) is used in many approaches to ensure UML diagrams consistency as shown in Table 1 and Table 2. OCL is considered as a direct approach in checking the consistency such as it is integrated in some modelling tools. Some approaches to ensure the consistency between UML diagrams using OCL are proposed in [24], [25], [33], [35], [37], [55]. In Briand *et al.* [12], [13], an automatic change impact analysis is developed to detect the changes between two different versions of a UML model automatically. The UML model is composed of class, sequence, and statechart diagrams. In addition, consistency rules, which are formalized using OCL, are proposed. Horizontal and vertical traceability analyses are supported in this approach. This approach is concerned with keeping the software models in a consistent state and synchronized with the underlying source code [23]. Approaches in [12], [13], [56], [57] summarized the set of consistency rules between UML diagrams from the literature. In Egyed [24], [25], [58], [59], the change impact scope is determined based on a set of proposed consistency rules for UML class, sequence, and state diagrams. UML/Analyzer and Model/Analyzer tools [59] are developed to automate and evaluate the approach. A novel approach for improving the performance of incremental consistency checking was proposed in Reder and Egyed [32]. The basic idea of this approach is to focus on the parts that are affected by model changes and not to validate design rules in their entirety. The purpose of the research in Ali, Boufares, and Abdellatif [33] is to emphasize the importance of constraints global coherence in order to ensure the validity of the conceptual model (class diagram) at the design stage. This research classifies the integrity constraints that can be held in UML class diagrams at the conceptual perspective. Some of these constraints are OCL constraints, intra association constraints, and inter classes' constraints (generalization, compositions, and functional dependencies constraint). Standard OCL does not allow making changes to the model elements to resolve them [1]. CPNs can be used for checking and verifying UML model associated with OCL to check whether it meets the user requirement or not [60].

3.2. Transformational Approaches

The coevolution of OO software design and implementation approach is proposed in D'Hondt, De Volder, Mens, and Wuyts [61] and Wuyts [62]. The logic meta-programming is proposed as a way to affect a bi-directional link between software design and implementation. Automated coevolution of models using traceability analysis based on model transformation to code is proposed in Amar *et al.* [17]. A coevolution approach between a component-based architecture model and OO source code is proposed in Langhammer [5]. The coevolution in this approach is based on bidirectional mapping rules between architecture model and source code. Approaches in (García, Diaz, and Azanza [14], Cicchetti, Di Ruscio, Eramo, and Pierantoni [63], Wachsmuth [64], and Hößler, Soden, and Eichler [65]) discuss the coevolution between meta-models and models based on models transformation to meta-models. In these approaches, new updates are stored in a new version from the meta-model. According to Protic [15], model coevolution describes the problem of adapting models when their meta-models evolve.

Tracing model changes through model synchronization approach is proposed in Ivkovic *et al.* [66] to achieve traceability consistency. In this approach, models are transformed to use a graph meta-model. The transformed meta-model is then used to code model dependencies and equivalence relations are used to evaluate model synchronization. A change in a model is viewed as a combination of graph changes. A graph transformation approach is defined in Fryz and Kotulski [67] to check the consistency between use case and class diagrams. In Mens *et al.* [20], horizontal and evolutionary consistency rules between the UML class, sequence, and statechart diagrams are classified. In addition, the authors describe an extension to the UML meta-model to support the UML diagrams version. The authors discuss the importance of traceability

analysis and change propagation in UML diagrams but they provide no support for this [68]. A tool for synchronous refactoring of UML activity diagrams using model-to-model transformations is provided in [69]. Refactoring is applied to improve the internal structure of source code. A meta-model and a generic typology of operators is used to express different kinds of evolution. Malabarba, Pandey, Gragg, Barr, and Barnes [48] focuses specifically on dynamic Java. The authors extend the default Java class loader in such a way that class definitions can be replaced and objects or dependent classes can be updated. The replacement is initiated by the user through explicit calls to the class loader in the application program. According to [70], the graph transformation technique limited to check the structural inconsistencies only because it detect and resolve the inconsistencies which can be expressed as a graph structure only. Other approached in consistency and coevolution base on transformational models are provided in [71]-[74].

3.3. Formal Semantics Approaches

In this subsection, some approaches that develop a formal semantic in order to ensure the consistency and correctness of UML diagrams are discussed. Additionally, this research provides a complete study about formal approaches using CPNs to check the consistency and correctness of UML diagrams in [75], [76]. A complete survey of the UML diagrams' change impact analysis techniques are discussed in Lucas, Molina, and Toval [8]. One of the results of this survey is: formal languages are highly used to support detecting and determining the consistency and change impact between software models. A CPNs approach to check the sequence diagrams consistency with the system requirements is presented in [77]. In this approach, a technique for sequence diagrams to Petri Nets (PNs) transformation is presented for the purpose of requirements validation and verification. Shinkawa [7] approach requires a transformation from UML diagrams to other notations (CPNs) before checking the consistency. A framework for the verification of UML behavioural diagrams using PNs is proposed in Guerra and de Lara [78]. UML state charts, activity, and collaboration diagrams are transformed to PNs for verification. In Gongzheng and Guangquan [38], XYZ/E formal language [39] which is based on temporal logic is used to checking the consistency between state chart and sequence diagrams in UML 2.0. Formal approach graph grammars for checking the consistency of UML class and sequence diagrams are proposed in [79]. In Rajabi and Lee [80]-[82], a change management framework to support runtime changes in business process modelling languages is proposed (where the business processes are modelled using UML diagrams). The new changes are the result of creating, deleting, or modifying business processes. Object-Z and CSP formal languages [83] are used to check the consistency between UML class and state chart diagrams. B formal method is focused on refinement to code in checking the consistency between UML diagrams [84], where the refinement means "describes the new definitions of some parts of the specification's elements according to the required changes" [84]. Formal approaches are widely used in describing the behavioural of the UML diagrams using the executional models capability provided in CPNs.

- **Coloured Petri Nets**

UML is a powerful means for describing the static and dynamic aspects of systems, but remains semi-formal and lacks techniques for model validation and verification [85]. According to Lucas [8], formal specifications and mathematical foundations such as Coloured Petri Nets (CPNs) are widely used in handling of inconsistency problems among models and to automatically validate and verify the model dynamic behaviour. Due to the advantages offered by formal languages, the integration between UML and formal languages is recommended to solve the inconsistencies between UML diagrams [8]. The advantages from the integration of UML and CPNs are better representation of a system's complexity as well as ease in adapting, correcting, analysing, and reusing a model. Transformation rules are required to transform UML diagrams elements to CPNs. Approaches discussed in the literature on the transformation of UML diagrams

to CPNs focus on the part of UML diagrams, in particular, the behavioural diagrams. Additionally, the consistency check is based on a set of rules applied on CPNs model. The integration of UML and CPNs in supporting software models coevolution and consistency check is approached in by much research in solving the coevolution and inconsistency problem in UML diagrams.

3.4. Knowledge Representation Approaches

knowledge-based approaches for OO diagrams consistency check are discussed in (Calì, Calvanese, De Giacomo, and Lenzerini [86] , Baader [87], and Bolloju *et al.* [16]. Knowledge-based system methodology to verify the consistency of a given object model against a set of use cases (defined as a natural language narrative) is proposed in [16]. In this methodology, missing and invalid diagram elements are determined to help the analyst in creating consistent object models with the requirements identified in the use cases' narratives. Use case driven based rules in ensuring consistency of UML model approach was proposed in Ibrahim *et al.* [34]. In Van Der Straeten, Mens, Simmonds, and Jonckers [88], UML metamodel and user-defined models were transformed into descriptive logic to check for consistency.

3.5. UML Diagramming Tools Support

A case study is performed in Amba [89] to evaluate four management tools (IBM Rational RequisitePro, Borland CaliberRM, TopTeam Analyst, and Telelogic DOORS) in supporting change impact and traceability analysis. This study indicates all these tools have poor impact analysis features. This shows that impact analysis in these management tools is very limited and thus more effective methods are needed. Some UML diagramming tools such as Visual Paradigm tool detect the impact analysis based on the physical connection between diagrams' elements. Visual paradigm tool analyses the connection between the diagrams' elements based on the user selection for the dependency between the diagrams. ArgoUML detects incremental consistency check in UML diagrams but it requires annotated consistency rules [24]. A set of consistency check rules between UML diagrams are identified in [4]. These rules are helpful for developers in checking the consistency between class, activity, state chart, sequence, and communication diagrams. The research discussed the methods of applying these consistency rules. These methods are: manual, compulsory restriction, automatic maintenance, and dynamic check.

4. Discussion

Software change is inevitable in software project lifecycle. When new changes are applied to software, they would be having some impacts and inconsistencies with other parts of the original software. Nowadays, effective change management is essential for organisational development and survival in order to keep track of changes and to reduce risks and costs. Change management has been recognized as "the most difficult, costly and labour-intensive activity in the software development life cycle" Li *et al.* [26]. One of the main issues in software change management is to detect and resolve the coevolution among software artefacts to determine the change impact and change propagation. Detecting and resolving the coevolution among software models is of tremendous significance for the field of software design and development to assess the change consequences. Software models are highly dynamic and evolve from requirements through implementation [66]. It is important to investigate how to integrate software changes into software models [90], [91]. A change management technique is required to support the criteria of flexibility, adaptability, and dynamic reaction to changes in software models.

OO modelling is widely used in software analysis and design. It describes a system by modelling different perspectives using its structural, behavioural, and interaction diagrams. UML defines different diagrams relations between these diagrams are complex, and may lead to inconsistent UML diagrams [4], [92]. Coevolution among different perspectives or views of UML diagrams means that the modification in one

diagram should be reflected to other related diagrams to ensure the consistency of all diagrams. If the effect of changes in UML diagrams is not addressed adequately among diagrams, it will result in further defects and decrease the maintainability, and increased gaps between high-level design and implementation [2], [9], [23], [34].

4.1. UML Diagrams Change Effects

In this section, we summarized the set of diagrams or diagram elements affected by the change. This can also called the cost of the change. The higher impacted diagrams and elements, the more severe the change. As shown in the figures and tables provided in this section, the results show how strong the relation among the class diagram and other models. This explains the large number of change impact templates and patterns proposed for the class diagram. The change impact for the diagrams' elements can be defined based on the dependency relations; some examples for these relations are the following:

- ∃ *e*(diagram element) ∈ CD: If (*e* is changed) Then (all diagrams are affected)

Classes, attributes, and operations in the class diagram are used or invoked in all UML diagrams.

- ∃ *e* ∈ OD: If (e is changed) Then (all diagrams are affected except the CD)

Objects are used in the structural, behavioral, and interaction diagrams

- ∃ *e* ∈ CoD : If (*e* is changed) Then (DD is affected)

CoD and DD are dependent on each other; the change in one of them will affect the other.

- ∃ *e* ∈ DD: If (*e* is changed) Then (CoD is affected)
- ∃ *e* ∈ UCD: If (*e* is changed) Then (AD, SCD, SD, CommD, TD, and IOD are affected)

The dynamic behavior of the UCD is described using the AD, SCD, SD, and CommD. The flow of control in the AD is from activity to activity. The flow of control in the SD and CommD is from object to object. TD and IOD are affected indirectly by the change in the UCD because their elements are derived from the AD and interaction diagrams.

- ∃ *e* ∈ AD: If (*e* is changed) Then (UCD, SCD, SD, CommD, IOD, and TD are affected)

An AD represents the internal behavior of the CD, UCD, and SCD. IOD and TD elements are derived from the AD elements, in addition to interaction elements added in the IOD. The AD shows how those activities depend on one another.

- ∃ *e* ∈ SCD: If (*e* is changed) Then (UCD,AD, SD, CommD, TD, and IOD are affected)

Dynamic behavior of the SCD is described using the AD, SD, and CommD. TD and IOD are affected indirectly by the SCD changes because their elements are derived from the AD and interaction diagrams.

- ∃ *e* ∈ SD: If (*e* is changed) Then (UCD, AD, SCD, CommD, and IOD are affected)
- ∃ *e* ∈ CommD: If (*e* is changed) Then (UCD, AD, SCD, SD, and IOD are affected)
- ∃ *e* ∈ PD, CSD, IOD, and TD: If (*e* is changed) Then (No diagrams are affected)

Table 3 summarizes the change effect on diagrams and elements for each element. This table also summarize the shared elements between the diagrams. These shared elements represent the relationships between templates. Where CD: Class Diagram, OD: Object Diagram, PD: Package Diagram, CoD: Component Diagram, DD: Deployment Diagram, CSD: Composite Structure Diagram, UCD: Use Case Diagram, AD: Activity Diagram, SCD: Statechart Diagram, SD: Sequence Diagram, CommD: Communication Diagram, TD: Timing Diagram, and IOD: Interaction Overview Diagram.

Information about the number of affected diagrams by updating each UML diagram and the number of update operation supported are summarized in Table 4, Fig. 1, and Fig. 2. Self, direct, and indirect dependencies are considered. More information about the dependency between diagrams (change effect between diagrams) are provided in Fig. 3.

Table 3. The Change Effect on Diagrams Elements

Template # / Diagram	Affected Elements
CD Attribute Changes	OD Object States, AD Object States, SCD variables, SD Object States, CommD Object States
CD Operation Change	OD Object States, CoD component operation, DD component operation, UCD Use case, AD Activities and Sub Activities, Actions, SCD Events, SD Sequence diagrams states, Messages, CommD Messages
CD Class Changes	OD, AD, SD, CommD Object States
CD Association Changes	AD Seq. of Activities, cntrl node, call behaviour, SD operators
CD Navigability Arrow Changes	OD Object Flow, AD Object and Control Flow, SD Object Flow
CD Dependency Changes	PD ,Cod ,DD dependency
OD Object (Class instance) Changes	OD, AD, SD, CommD Object Instances
OD Object States Changes	OD, AD, SD, CommD Object States
CoD and DD Node Changes	CoD, DD component operation Node
CoD and DD Component Operation Changes	CoD, DD component operation
CoD and DD Dependency Changes	CoD, DD dependency relation
UCD Communication (association) Changes	SD, CommD Objects Links
UCD Extend/Include/Generalize/Use Relations Changes	AD Seq. of Activities, cntrl node, call behaviour, SD operators
UCD Use case Description Change	AD sequence of Activities
AD Sub-Activity/SCD Activity Changes	AD Activity/, Sub-Activity, SCD Event, SD Operators
UCD, SCD, and AD Action Changes	UCD, AD, SCD Action, SD Operators
AD Control Flow Changes	AD, SD, CommD Object Flow
AD Object Flow Changes	SD, CommD Object/control Flow
AD Control Nodes (Fork, Join, Merge, and Decision) Changes	UCD relationships, AD Control, Nodes, SD operators
AD Activity Sequence Changes	UCD description
AD, SD, and CommD Iteration /Loop Changes	AD, SCD, SD, CommD, Loop/ branches
SCD, AD, and SD Guard Condition Changes	UCD, AD, SCD, SD, CommD, IOD Guard
SCD Composite State and Sub-State Changes	SD message passing CommD message passing
SD and CommD Object Changes	Same as in OD Object changes
SD Message Changes	CD Comm method dynamic binding, SD synchronous asynchronous messages, CommD synchronous asynchronous messages
SD Synchronous and Asynchronous Messages Changes	CD Comm method dynamic binding CoD interface, DD interface, SCD events, SD synchronous asynchronous messages, CommD synchronous asynchronous messages
SD Operators (alt/ opt / ref / par) Changes	UCD Description, relationships, AD Cntrl node branches
SD Action Bars/Lifelines Changes	AD Activity sequence

Table 4. Statistics for Updating UML Diagram Elements

Diagram Name	**No of Affected Diagrams**	**# of Update Operations**
Class Diagram	13	41
Object Diagram	12	9
Package Diagram	1	5
Component Diagram and Deployment Diagram	2	13

Diagram Name	No of Affected Diagrams	# of Update Operations
Composite Structure Diagram	1	6
Use Case Diagram	7	12
Activity Diagram	7	17
State Chart Diagram	7	18
Sequence Diagram	6	23
Communication Diagram	6	17
Interaction Overview Diagram	1	3
Timing Diagram	1	5

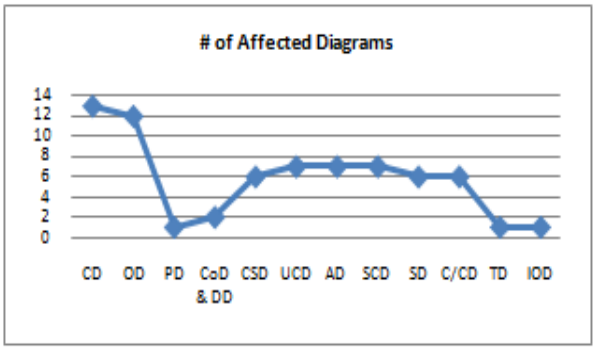

Fig. 1. Number of update operations supported by each UML diagrams.

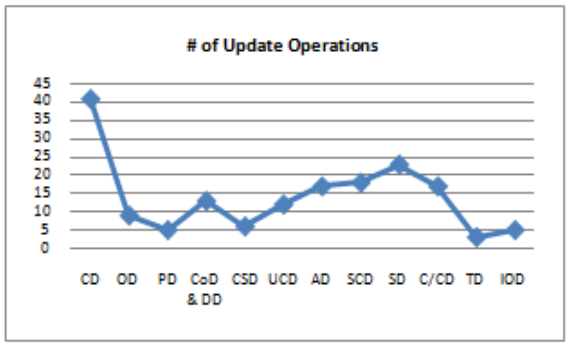

Fig. 2. Number of diagrams affected by updating UML diagram.

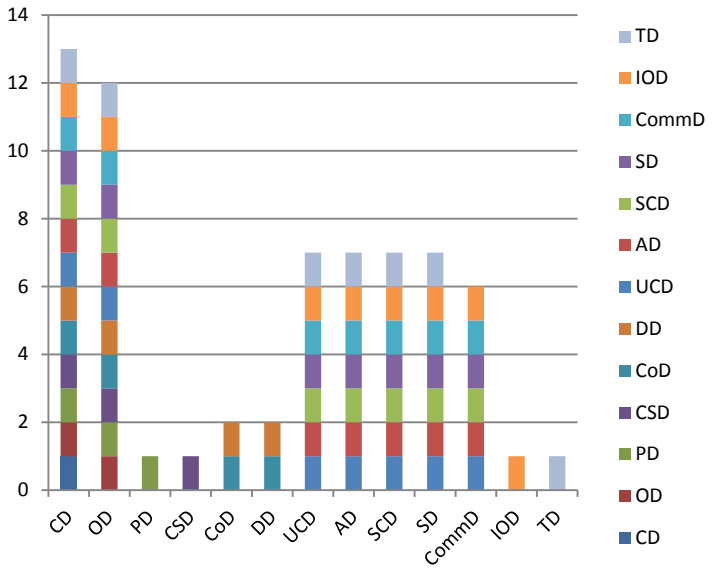

Fig. 3. Diagrams dependency/change effect.

4.2. Research Issues

Providing a change management technique to manage the coevolution among software models is one of the popular research areas in software analysis and design due to their numerous applications, and to ensure the models correctness in response to changes on them [93]. Solving the coevolution and inconsistency problems in software models especially UML diagrams is a highly active research in which a considerable research work has been done [3], [9], [10]. However, there are important gaps and limitations still open for research. One of these limitations is the management of inconsistency problem in adding or changing or deleting new models or diagrams or diagram elements. In addition, although the previous

approaches in the state-of-the-art research provide solutions to handle software changes in UML diagrams, these approaches are concerned with some of the UML diagrams (i.e. the class, sequence, and statechart diagrams) and concentrate on checking the consistency by comparing two different versions from the same model. There is a need to handle software model change management not only the consistency check comprehensively using all UML structural, behavioural, and interaction diagrams including the diagrams relations and checking the integrity and consistency between diagrams. As a summary of the main issues from the state of the art to be addressed:

- Software models are highly dynamic and evolve from requirements through implementation. In order to respond quickly to varying requirements, it is extremely important to provide a change management technique to keep track of changes and to realise flexible and consistent software models.
- An OO modelling language describes a system by modelling different perspectives using its structural, behavioural, and interaction diagrams. The coevolution among these diagrams is high; therefore it is crucial to check the coevolution between the perspectives in these diagrams in order to control the change and keep these different views or perspectives consistent.
- Provides a new structure for the integration between OO and CPNs to support model changes. UML as a standard language for modelling OO software systems is a semi-formal language and does not automatically support validation and verification of the coevolution between software models. Integrating with CPNs will help in automatic validation and verification. Additionally this will increases the structuring capabilities of CPNs to include an OO structure.

Hence, it is a challenge to address the coevolution and inconsistency problems in UML diagrams to support the coevolution between UML diagrams and to keep track of changes in UML diagrams. This includes ensuring the consistency between UML diagrams, tracing the diagrams' dependency, and determining the effect of the change in these diagrams after each change operation. Coping with software changes is one of the major issues in software analysis and design. Therefore, this research introduces the challenges to cover the limitations about coevolution and consistency of UML diagrams.

Improving the effectiveness and the accuracy of the state-of-the-art coevolution techniques in managing OO diagrams changes is an important issue and still much work needed to be done to provide the flexibility, adaptability, and dynamic reaction to changes comprehensively.

5. Conclusion and Future Work

As software evolves, analysis and design models should be modified, correspondingly. Coevolution between diagrams involves both impact analysis and change propagation. This research defined and discussed the current approach in UML diagrams coevolution and consistency. Additionally this research defined some research issues needed to be addressed in UML diagrams changes. As a result for this survey, to cope with the changes in the software process, a novel approach for coevolution framework is required to manipulate the change effect in the UML diagrams' elements. UML diagrams are modeled from different perspectives using UML structural, behavioral, and interaction diagrams, detecting the elements affected by the change in systems design modeled using UML diagrams is required. This include controlling the evolution of UML diagrams by identifying and managing the model changes, ensuring the correctness and consistency of models, the impact of changes based on the relationships between diagrams, and performance analysis. This can be considered as a future work for this research.

References

[1] Khalil, A., & Dingel, J. (2013). *Supporting the Evolution of UML Models in Model Driven Software*

Development: A Survey (Technical Report No. 2013-602). School of Computing, Queen's University.

[2] Lehnert, S., & Riebisch, M. (2013). Rule-based impact analysis for heterogeneous software artifacts. *Proceedings of IEEE 17th European Conference on Software Maintenance and Reengineering (CSMR)*.

[3] Puissant, J. P., Straeten, R. V. D., & Mens, T. (2013). Resolving model inconsistencies using automated regression planning. *Software & Systems Modeling*, 1-21.

[4] Liu, X. (2013). Identification and check of inconsistencies between UML diagrams. *Journal of Software Engineering and Applications, 6*, 73-77.

[5] Langhammer, M. (2013). Co-evolution of component-based architecture-model and object-oriented source code. *Proceedings of the 18th International Doctoral Symposium on Components and Architecture*. ACM.

[6] Breivold, H.P., Crnkovic, I., & Larsson, M. (2012). A systematic review of software architecture evolution research. *Information and Software Technology, 54(1)*, 16-40.

[7] Shinkawa, Y. (2006). Inter-model consistency in UML based on CPN formalism. *Proceedings of 13th Asia Pacific Software Engineering Conference (APSEC 2006)*. IEEE.

[8] Lucas, F.J., Molina, F., & Toval, A. (2009). A systematic review of UML model consistency management. *Information and Software Technology, 51(12)*, 1631-1645.

[9] Puczynski, P. J. (2012). *Checking Consistency between Interaction Diagrams and State Machines in UML Models*. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark.

[10] Dubauskaite, R., & Vasilecas, O. (2013). Method on specifying consistency rules among different aspect models, expressed in UML. *Electronics and Electrical Engineering, 19(3)*, 77-81.

[11] Spanoudakis, G., & Zisman, A. (2001). Inconsistency management in software engineering: Survey and open research issues. *Handbook of Software Engineering and Knowledge Engineering, 1*, 329-380.

[12] Briand, L. C., Labiche, Y., & O'sullivan, L. (2003). Impact analysis and change management of UML models. *Proceedings of International Conference on Software Maintenance (ICSM 2003)*.

[13] Briand, L. C., Labiche, Y., & Yue, T. (2009). Automated traceability analysis for UML model refinements. *Information and Software Technology, 51(2)*, 512-527.

[14] García, J., Diaz, O., & Azanza, M. (2013). Model transformation co-evolution: A semi-automatic approach. *Software Language Engineering*, 144-163. Springer.

[15] Protic, Z. (2011). *Configuration Management for Models: Generic Methods for Model Comparison and Model Co-evolution*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

[16] Bolloju, N., Schneider, C., & Sugumaran, V. (2012). A knowledge-based system for improving the consistency between object models and use case narratives. *Expert Systems with Applications, 39(10)*.

[17] Amar, B., *et al.* Automatic co-evolution of models using traceability. (2013). *Software and Data Technologies,* 125-139. Springer.

[18] Bohner, S. A. Software change impacts-an evolving perspective. (2002). *Proceedings of IEEE International Conference on Software Maintenance.*

[19] Kagdi, H., Gethers, M. & Poshyvanyk, D. (2012). Integrating conceptual and logical couplings for change impact analysis in software. *Empirical Software Engineering*, 1-37.

[20] Mens, T., Straeten, R. V. D., & Simmonds, J. (2005). A framework for managing consistency of evolving UML models. *Software Evolution with UML and XML*, 1-31.

[21] Mahmood, Z. & Mahmood, R. B. T. (2015). Category, strategy and validation of software change impact analysis. *International Journal Of Engineering And Computer Science, 11(4)*, 11126-11128.

[22] Chen, C.-Y., She, C.-W., & Tang, J.-D. (2007). An object-based, attribute-oriented approach for software change impact analysis. *Proceedings of IEEE International Conference on Industrial Engineering and Engineering Management*.

[23] Lehnert, S. (2011). A review of software change impact analysis (Tech. Rep. No. 39). Ilmenau University of Technology.

[24] Egyed, A. (2006). Instant consistency checking for the UML. *Proceedings of the 28th International Conference on Software Engineering*. ACM.

[25] Egyed, A. (2011). Automatically detecting and tracking inconsistencies in software design models. *IEEE Transactions on Software Engineering, 37(2)*, 188-204.

[26] Li, B., *et al.* (2012). A survey of code-based change impact analysis techniques. *Software Testing, Verification and Reliability*.

[27] Stephan, M., & Cordy, J. R. (2013). A survey of model comparison approaches and applications. *Modelsward*, 265-277.

[28] Mohan, K., *et al.* (2008). Improving change management in software development: Integrating traceability and software configuration management. *Decision Support Systems, 45(4)*, 922-936.

[29] Park, S., Kim, H., & Bae, D. H. (2009). Change impact analysis of a software process using process slicing. *Proceedings of 9th International Conference on Quality Software (QSIC'09).* IEEE.

[30] Lucia, A. D., Fasano, F., & Oliveto, R. (2008). Traceability management for impact analysis. *Proceedings of Frontiers of Software Maintenance, FoSM 2008.* IEEE.

[31] Ekanayake, E., & Kodituwakku, S. R. (2015). Consistency checking of UML class and sequence diagrams. *Proceedings of 8th International Conference on Ubi-Media Computing (UMEDIA)*. IEEE.

[32] Reder, A., & Egyed, A. (2012). Incremental consistency checking for complex design rules and larger model changes. *Proceedings of International Conference on Model Driven Engineering Languages and Systems* (pp. 202-218). Springer.

[33] Ali, A., Boufares, F., & Abdellatif, A. (2006). Checking constraints consistency in UML class diagrams. *Proceedings of 2nd Information and Communication Technologies, ICTTA'06*. IEEE.

[34] Ibrahim, N., *et al.* (2013). Use case driven based rules in ensuring consistency of UML model. *Global Journal on Technology*, 1.

[35] Elaasar, M., & Briand, L. (2004). An overview of UML consistency management (Technical Report No. SCE-04-18). Carleton University, Canada.

[36] Reder, A., & Egyed, A. (2013). Determining the cause of a design model inconsistency. *IEEE Transactions on Software Engineering,* 1.

[37] Millan, T., *et al.* An OCL extension for checking and transforming UML models. (2009). *Proceedings of International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS'09)*. Cambridge, United Kingdom.

[38] Gongzheng, L., & Guangquan, Z. (2010). An approach to check the consistency between the UML 2.0 dynamic diagrams. *Proceedings of 5th International Conference on Computer Science and Education (ICCSE)*. IEEE.

[39] Tang, Z. (2002). Temporal logic programming and software engineering. *VCI, 1(1, 2),* 5. Bering: Science Press.

[40] Pnueli, A. (1977). The temporal logic of programs. *Proceedings of 18th Annual Symposium on Foundations of Computer Science*. IEEE.

[41] Isaac, S., & Navon, R. (2013). A graph-based model for the identification of the impact of design changes. *Automation in Construction, 31*, 31-40.

[42] Weiser, M. Program slicing. (1984). *IEEE Transactions on Software Engineering, 1984(4)*, 352-357.

[43] Zhao, J. (2002). Change impact analysis for aspect-oriented software evolution. *Proceedings of the International Workshop on Principles of Software Evolution.* ACM.

[44] Bishop, L. (2004). Incremental impact analysis for object-oriented software. Lowa State University.

[45] Xing, Z., & Stroulia, E. (2005). Analyzing the evolutionary history of the logical design of object-oriented software. *IEEE Transactions on Software Engineering, 31(10)*, 850-868.

[46] Gethers, M., *et al.* (2012). Integrated impact analysis for managing software changes. *Proceedings of 34th International Conference on Software Engineering (ICSE).* IEEE.

[47] Costanza, P. (2001). Dynamic object replacement and implementation-only classes. *Proceedings of 6th International Workshop on Component-Oriented Programming (WCOP 2001) at ECOOP*.

[48] Malabarba, S., *et al.* (2000). Runtime support for type-safe dynamic Java classes. Springer.

[49] Vandewoude, Y., & Berbers, Y. (2002). Run-time evolution for embedded component-oriented systems. *Proceedings of International Conference on Software Maintenance*. IEEE.

[50] Huang, L., & Song, Y.-T. (2007). Precise dynamic impact analysis with dependency analysis for object-oriented programs. *Proceedings of 5th ACIS International Conference on Software Engineering Research, Management & Applications, SERA 2007*. IEEE.

[51] Sun, X., *et al.* (2010). Change impact analysis based on a taxonomy of change types. *Proceedings of IEEE 34th Annual Computer Software and Applications Conference (COMPSAC)*.

[52] Torchiano, M., & Ricca, F. (2010). Impact analysis by means of unstructured knowledge in the context of bug repositories. *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*.

[53] Kung, D., *et al.* (1994). Change impact identification in object oriented software maintenance. *Proceedings of IEEE International Conference on Software Maintenance*..

[54] Zalewski, M., & Schupp, S. (2006). Change impact analysis for generic libraries. *Proceedings of 22nd IEEE International Conference on Software Maintenance, ICSM'06*.

[55] Vasilecas, O., Dubauskaitė, R., & Rupnik, R. (2011). Consistency checking of UML business model. *Technological and Economic Development of Economy, 17(1)*, 133-150.

[56] DAMIANO, T., LABICHE, Y., & GENERO, M. (2015). *A Systematic Identification of Consistency Rules for UML Diagrams* (Technical Report No. SCE-15-01). Carleton University.

[57] Torre, D. (2015). On validating UML consistency rules. *Proceedings of IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*.

[58] Egyed, A. (2007). Fixing inconsistencies in UML design models. *Proceedings of 29th International Conference on Software Engineering, ICSE 2007a*. IEEE.

[59] Egyed, A. (2007). Uml/analyzer: A tool for the instant consistency checking of uml models. *Proceedings of 29th International Conference on Software Engineering, ICSE 2007*b. IEEE.

[60] Sharaff, A. (2013). A methodology for validation of OCL constraints using coloured petri nets. *International Journal of Scientific & Engineering Research, 4(1)*.

[61] D'Hondt, T., *et al.* (2002). Co-evolution of object-oriented software design and implementation. *Software Architectures and Component Technology*, 207-224. Springer.

[62] Wuyts, R. (2001). A logic meta-programming approach to support the co-evolution of object-oriented design and implementation. PhD thesis, Vrije Universiteit Brussel.

[63] Cicchetti, A., *et al.* (2008). Automating co-evolution in model-driven engineering. *Proceedings of 12th International IEEE Enterprise Distributed Object Computing Conference. EDOC'08*.

[64] Wachsmuth, G. (2007). Metamodel adaptation and model co-adaptation. *Proceedings of ECOOP 2007–Object-Oriented Programming* (pp. 600-624). Springer.

[65] Hößler, J., Soden, M., & Eichler, H. (2005). Coevolution of models, metamodels and transformations. *Models and Human Reasoning. Wissenschaft and Technik Verlag*, 129-154. Berlin.

[66] Ivkovic, I., & Kontogiannis, K. (2004). Tracing evolution changes of software artifacts through model synchronization. *Proceedings of* 20th IEEE In ternational Conference on Software Maintenance.

[67] Fryz, L., & Kotulski, L. (2007). Assurance of system consistency during independent creation of UML diagrams. *Proceedings of 2nd International Conference on Dependability of Computer Systems: DepCoS-RELCOMEX'07*. IEEE.

[68] Herzig, S., *et al.* (2011). A conceptual framework for consistency management in model-based systems engineering. *Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2011*. ASME.

[69] Einarsson, H. ó., & Neukirchen, H. (2012). An approach and tool for synchronous refactoring of UML diagrams and models using model-to-model transformations. *Proceedings of the Fifth Workshop on Refactoring Tools*. ACM.

[70] Puissant, J. P. (2012). *Resolving Inconsistencies in Model-Driven Engineering using Automated Planning*. PhD thesis, Universit de Mons.

[71] Khan, A. H. & Porres, I. (2015). Consistency of UML class, object and statechart diagrams using ontology reasoners. *Journal of Visual Languages & Computing, 26*, 42-65.

[72] Kusel, A., *et al.* (2015). Systematic Co-evolution of OCL expressions. *Proceedings of 11th APCCM*: *Vol. 27*. (p. 30).

[73] Demuth, A., *et al.* (2016). Co-evolution of metamodels and models through consistent change propagation. *Journal of Systems and Software, 111*, 281-297.

[74] Dang, D.-H., & Gogolla, M. (2016). An OCL-Based Framework for Model Transformations. *VNU Journal of Science: Computer Science and Communication Engineering, 32(1)*.

[75] Rajabi, B. A., & Lee, S. P. (2009). A study of the software tools capabilities in translating UML models to PN models. *International Journal of Intelligent Information Technology Application (IJIITA), 2(5)*, 224-228.

[76] Rajabi, B. A., & Lee, S. P. (2014). Consistent integration between object oriented and coloured petri nets models. *The International Arab Journal of Information Technology, 11(4)*.

[77] Ouardani, A., *et al.* (2006). A Meta-modeling approach for sequence diagrams to petri nets transformation within the requirements validation process. *Proceedings of the European Simulation and Modeling Conference*.

[78] Guerra, E., & Lara, J. (2003). A framework for the verification of UML models: Examples using petri nets. *Proceedings of JISBD*.

[79] Tsiolakis, A., & Ehrig, H. (2000). Consistency analysis of UML class and sequence diagrams using attributed graph grammars. *Proceedings of Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems.* Berlin.

[80] Rajabi, B. A., & Lee, S. P. (2009). Change management in business process modeling based on object oriented petri net. *Proceedings of World Academy of Science, Engineering and Technology, 38*, 12-17.

[81] Rajabi, B. A., & Lee, S. P. (2009).Runtime change management based on object oriented petri net. *Proceedings of International Conference on Information Management and Engineering, ICIME'09*. IEEE.

[82] Rajabi, B. A., & Lee, S. P. (2010). Modeling and analysis of change management in dynamic business process. *International Journal of Computer and Electrical Engineering, 2(1)*, 181-189.

[83] Rasch, H., & Wehrheim, H. (2003). Checking consistency in UML diagrams: Classes and state machines, in formal methods for open object-based distributed systems. *Springer*, 229-243.

[84] Ossami, D. D. O., Jacquot, J.-P. & Souquières, J. (2005). Consistency in UML and B multi-view specifications. *Integrated Formal Methods*, 386-405.

[85] Niepostyn, S. J. (2015). The sufficient criteria for consistent modelling of the use case realization diagrams with a new functional-structure-behaviour UML diagram. *Przegląd Elektrotechniczny Sigma NOT, 2015(2)*, 31-35.

[86] Calì, A., *et al.* (2002). A formal framework for reasoning on UML class diagrams. *Proceedings of International Symposium on Foundations of Intelligent Systems* (pp. 503-513). Springer.

[87] Baader, F. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge university press.

[88] Straeten, R. V. D., *et al.* (2003). Using description logic to maintain consistency between UML models, in «UML» 2003-The Unified Modeling Language. *Modeling Languages and Applications*. Springer.

[89] Abma, B. (2009). *Evaluation of Requirements Management Tools with Support for Traceability-Based Change Impact Analysis*. Master's thesis, University of Twente.

[90] April, A., & Abran, A. (2012). Software maintenance management: Evaluation and continuous improvement. Wiley-IEEE Computer Society Press.

[91] Mens, T., *et al.* (2005). Challenges in software evolution. *Proceedings of Eighth International Workshop on Principles of Software Evolution*. IEEE.

[92] Torre, D., Labiche, Y., & Genero, M. (2014). UML consistency rules: A systematic mapping study. *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. ACM.

[93] Williams, B. J., & Carver, J. C. (2010). Characterizing software architecture changes: A systematic review. *Information and Software Technology, 52(1)*, 31-51.

Bassam Atieh Rajabi received his M.Sc. degree in computer science from Alquds University, Jerusalem, Palestine, in 2005. Currently, he is a PhD student in computer science at University of Malaya, Malaysia. From 2001 to 2004, he was a research and teaching assistant with the Computer Science Department, Alquds University-Palestine. He was a lecturer and dean assistant for administrative affairs from 2005 till now with Wajdi University College of Technology-Palestine. His areas of interest are software design and modeling techniques. He has been an active member in the reviewer committees of several local and international conferences and journals.

Sai Peck Lee is a professor at Faculty of Computer Science and Information Technology, University of Malaya. She obtained her master of computer science from University of Malaya, her Diplôme d'Études Approfondies (D.E.A.) in computer science from Université Pierre et Marie Curie (Paris VI) and her Ph.D. degree in computer science from Université Panthéon-Sorbonne (Paris I). Her current research interests in software engineering include object-oriented techniques and case tools, software reuse, requirements engineering, application and persistence frameworks, information systems and database engineering. She has published an academic book, a few book chapters as well as more than 100 papers in various local and international conferences and journals. She has been an active member in the reviewer committees and programme committees of several local and international conferences. She is currently in several experts referee panels, both locally and internationally