Change Management Framework to Support UML Diagrams Changes

Bassam Rajabi and Sai Peck Lee
Faculty of Computer Science and Information Technology, University of Malaya, Malaysia

Abstract: *An effective change management technique is essential to keep track of changes and to ensure that software projects are implemented in the most effective way. Unified Modeling Language (UML) diagrams are widely adopted in software analysis and design. UML diagrams are divided into different perspectives in modelling a problem domain. Preserving the consistency among these diagrams is very crucial so that they can be updated continuously to reflect software changes. In this research, a change management framework is proposed to trace the dependency and to determine the effect of the change in UML diagrams incrementally after each update operation. A set of 45 change impact and traceability analysis templates for all types of change in UML diagrams elements are proposed to detect the change affected and to maintain the diagrams consistency and integrity. The proposed framework is modeled and simulated using Coloured Petri Nets (CPNs) formal language. UML is powerful in describing the static and dynamic aspects of systems, but remains semi-formal and lacks techniques for models validation and verification especially if these diagrams updated continuously. Formal specifications and mathematical foundations such as CPNs are used to automatically validate and verify the behavior of the model. A new structure is proposed for the mutual integration between UML and CPNs modeling languages to support model changes.*

Keywords: *Change impact, change management, traceability analysis, unified modeling language, coloured petri nets.*

Received April 16, 2016; accepted September 24, 2017

1. Introduction

Software modeling is one of the most important activities for large-scale software development. Software change is continuous and unavoidable due to rapidly changing requirements across software systems. Software change management is an essential activity in software project life cycle to ensure that the changes are implemented in the most effective way including maintaining the integrity and the traceability of the changes. Software models continues to face challenges in coping with dynamic business environments where requirements and goals are constantly changing at the runtime, and thus business users are demanding adaptive and flexible modeling techniques. A change impact and traceability analysis technique is required to determine the change effect and maintain the consistency and integrity of the software product. Change impact analysis is a critical issue in software project management. It is a systematic approach to understand the impacts of software changes. It is essential to identify the parts that require retesting, to reduce potential errors due to unknown change impacts, and to improve overall efficiency in software maintenance. The change impact can have local or global concerns. Local changes are concerned with instances; it is necessary to detect if they have indirect impacts on other instances. Global changes are concerned with process definitions. Traceability analysis is important in change impact. It is an analysis of the dependencies between and across software artifacts and actors at all levels of the software process.

The output of traceability analysis consists of all the components affected by the change. Change impact and traceability analysis techniques are classified into two categories [16]: code based techniques such as program slicing, and model based techniques such as program dependence graphs. Code based impact analysis techniques require the implementation details of a change request or a precise change implementation plan prior to determining change impacts. Model based impact analysis techniques identify and determine change impacts in the earlier phase without using program code, and make proper decisions before considering any change implementation details.

Object-Oriented (OO) modeling language is widely used in software analysis and design. Unified Modeling Language (UML), which is mainly based on the object-orientation, is powerful for describing the static and dynamic aspects of systems, but remains semi-formal and is lack of techniques for model validation and verification [19]. In addition, it is difficult to keep consistency between models presented by different UML diagrams. Since UML diagrams can be divided into different categories, where each category focuses on a different perspective of a problem domain, one of the critical issues is to keep consistency among diagrams [15]. The use of UML diagrams in modeling software systems leads to a large number of interdependent diagrams. It is necessary to preserve these diagrams' consistency and integrity since they are updated continuously to reflect the software model change [12]. UML is powerful in describing the static and dynamic aspects of systems, but remains semi-formal and lacks techniques for diagrams validation and verification especially if these diagrams updated continuously. Formal specifications and mathematical foundations such as Coloured Petri Nets (CPNs) are used to automatically validate and verify the behavior of the model. The main advantages expected from the integration of UML and CPNs modeling languages are: better representation of systems complexity as well as ease to adapt, correct, analyze, and reuse a model. This research proposed a change management framework to trace the dependency and to determine the effect of change of UML diagrams elements incrementally; it is not only a comparison between two versions from the same diagram, also it is used to check the consistency, impact, and traceability after creating, deleting, or modifying any diagram element by applying the same idea of syntax checking incrementally in CPNs models. In addition, change impact and traceability analysis templates are proposed for all types of change in the UML diagrams. These templates define information about the types of change supported for each diagram, information on change impact, dependency between diagrams, and rules to maintain the integrity and consistency between diagrams. The proposed change management framework is modeled and simulated using CPN Tools. This research proposes a new structure for the mutual integration between UML and CPNs modeling languages to support model changes, the new structure include set of rules to check and maintain the consistency and integrity based on UML diagrams relations. The rest of the paper is organized as follows: In Section 2, an overview and related works of change impact, traceability analysis, and integration between UML and CPNs models are discussed. In section 3, the proposed framework is explained and discussed. In sections 4 and 5, we discuss, analyze, and summarize the research findings and future work.

2. Related Works

In this section, approaches related to the change management frameworks are discussed. Software configuration management is concerned with managing the evolving software systems. Model based techniques identify change impacts by tracking the dependencies of software objects and classes. Dependencies among artifacts refer to how changes in some artifacts may impact other artifacts [18]. Traceability and consistency types are discussed in [7, 17] Vertical traceability refers to the ability to trace dependent artifacts within a model, while horizontal traceability refers to the ability to trace artifacts between different models within the same version. Evolutionary traceability indicates the consistency between different versions of the same model. In [5], an approach for performing change impact analysis is presented to describe changeable items (objects, attributes, and linkage) and their relations. Horizontal and evolutionary consistency between the UML class, sequence, and statechart diagrams are discussed in [17]. A case study is performed in [1] to evaluate four requirements 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 current requirements management tools is very limited and more effective methods are needed. Decades of research efforts have produced a wide spectrum of approaches and techniques in checking the change impact and inconsistency among OO diagrams. Some of these approaches can be classified into direct, transformational, formal semantics approaches [23]. Direct approaches use the constructs of OO and Object Constraints Language (OCL) [3, 4, 6]. Standard OCL does not allow making changes to the model elements to resolve them [13]. CPNs can be used for checking and verifying UML model associated with OCL to check whether it meets the user requirement or not [24]. Briand *et al*. [3] and Briand *et al*. [4] approach is concerned with keeping the software models in a consistent state and synchronized with the underlying source code [14]. In Egyed [8, 9, 10, 11], 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. Transformational approaches derive a common notation by transforming one model to another. The coevolution in this approach is based on bidirectional mapping rules between architecture model and source code. According to [20], 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. Formal approaches develop formal semantics for the OO diagrams, while knowledge representation approaches use description logics as a representation language [2]. Formal approaches are widely used in describing the behavioural of the UML diagrams using the executable models capability provided in CPNs. Transforming UML diagrams to formal modelling language such as CPN models is considered one of the most effective ways to solve the software performance evaluation problems. The integration between UML and CPNs approaches is based on the combination of the best characteristics of the CPNs and UML design methods. UML describes the static aspects of systems, while CPNs model system dynamics and behavioural aspects. The graphical representation and automated analysis techniques in CPN tools are used to aid in the understanding of formal specifications. The transformation approaches have certain weaknesses, such that each transformation approach uses only a subset of the UML diagrams, and most of these transformations are based on behavioural UML diagrams, these approached focused only on a comparison between two versions from the same model to check if there are differences between them. To react to changes in its environment in a quick and flexible way, and to provide life cycle support of software change management, a framework for managing dynamic software changes is required. Although the previous approaches in the related works provide the earlier phase solution to handle software changes in UML diagrams, these approaches are concerned with only part of the UML diagrams (i.e., the class, sequence, and statechart diagrams). There is a need to handle software change and change impact comprehensively using all UML structural, behavioural, and interaction diagrams including the diagrams' relations and checking the integrity and consistency between diagrams. Also, most of these approaches are based on the comparison between two versions of the same diagram. Integration between UML and CPNs are discussed in our previouse work in [21]. Improving the effectiveness and the accuracy of the state-of-the-art 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.

3. Proposed Change Management Framework for OO Diagrams

In this research, a change management framework for UML diagrams is proposed. UML diagrams are modeled using UML structural, behavioural, and interaction diagrams. The change in the UML diagram is the result of creating, deleting, or modifying diagram element. The proposed framework is a type of software configuration management technique. Impact and traceability analysis is important in order to identify the parts that require retesting, and to improve the overall efficiency in software change management techniques. The nature of the change could be corrective or evolutionary. Corrective changes are implemented to correct a design error. Evolutionary changes are required due to the redesign or reconfiguration of processes. These changes are represented by rules to be discussed in section 3.2. The change effect could be local if the change in one diagram does not impact on other diagrams, or a global change which is concerned with relations between diagrams. The main components for the proposed framework which are used to determine the change effect are explained in Figure 1 which includes the detailed components for the proposed change management framework.

3.1. Software Model

A complete model can be represented using UML diagrams. UML 2.3 supports a variety of diagrams, which allow the developers to model software systems from different perspectives using UML structural, behavioural, and interaction diagrams. UML diagrams are interrelated; some components for one diagram may be derived from other diagrams. For example, an activity diagram can be used to model an operation associated with a use case or a class diagram. Since UML diagrams can be divided into different categories, where each category focuses on a different perspective of a problem domain, one of the critical issues that needs to be addressed is the maintenance of consistency among diagrams [25]. Structural Diagrams are used to construct the information structure. Behavioural Diagrams show how a system operates. Interaction Diagrams can be considered as a subset of behavioural diagrams.

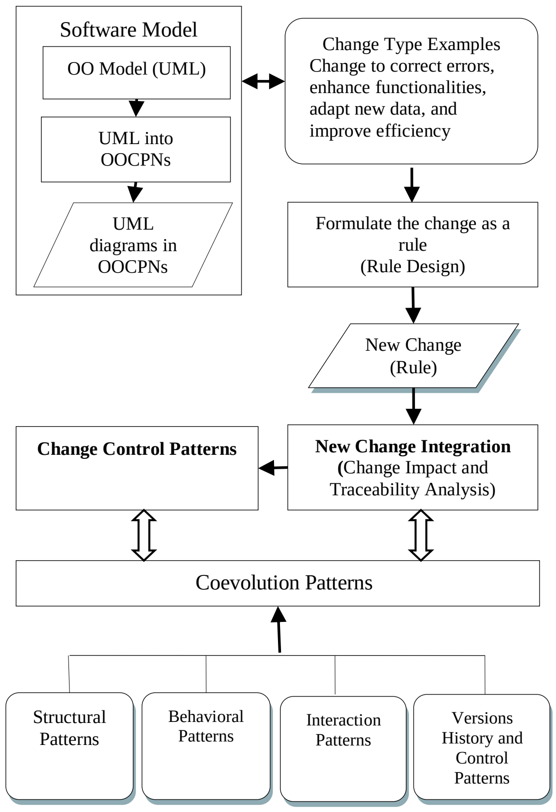

Software Model
OO Model (UML)
UML into OOCPNs
UML diagrams in OOCPNs
Change Type Examples
Change to correct errors, enhance functionalities, adapt new data, and improve efficiency
Formulate the change as a rule (Rule Design)
New Change (Rule)
Change Control Patterns
New Change Integration (Change Impact and Traceability Analysis)
Coevolution Patterns
Structural Patterns
Behavioral Patterns
Interaction Patterns
Versions History and Control Patterns

Figure 1. Proposed change management framework steps.

Transforming UML diagrams into executable models that are ready for analysis is significant, and providing an automated technique that can transform these diagrams into a mathematical model such as CPNs avoids redundancy in writing specifications. Many approaches for integrating OO modeling and PNs/CPNs have been investigated and developed. The transformation between UML diagrams and CPNs is partially supported for a subset of the UML diagrams as shown in [22]. This research focuses on the transformation of UML diagrams from the structural, behavioural, and interaction perspectives. This includes the new features in the UML modeling language such as the composite structure diagram, timing diagram, and interaction overview diagram. In addition, a new structure (Object Oriented Coloured Petri Nets (OOCPN)) to include rules to maintain the consistency and integrity is proposed to support model changes. The block diagram of the transformation process is shown in Figure 2.

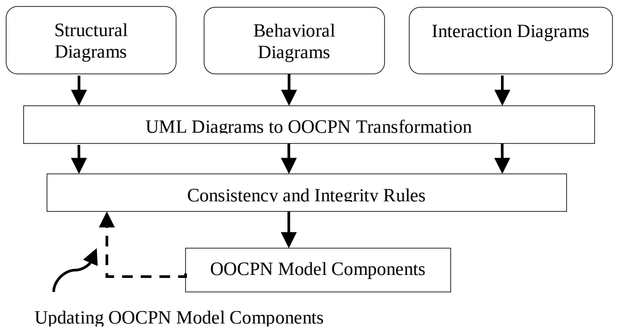

Structural Diagrams
Behavioral Diagrams
Interaction Diagrams
UML Diagrams to OOCPN Transformation
Consistency and Integrity Rules
OOCPN Model Components
Updating OOCPN Model Components

Figure 2. Block diagram for transforming UML diagrams to OOCPN model.

The components of UML structural, behavioural, and interaction diagrams are transformed to CPNs elements based on the proposed transformation rules. Consistency and integrity rules are checked during the transformation process and after updating the CPNs model.The general structure for the OOCPN model after the transformation of UML diagrams will be as follows: attributes and operations are transformed from the class diagram. These attributes and operations are used by other OOCPN model components. Classes are organized into subpages or subnets. These subpages can be instantiated using tokens which represent the objects. Related subpages are grouped together according to the package and composite structure diagrams. Objects behavior and interaction are described using the transformed behavioural and interaction diagrams. The state chart diagram describes the objects behavior by states and events. Activity diagram describes the flow of control form activity to activity. Sequence diagram describes the flow of control from object to object. Each activity could have starting and finishing time to determine the activities sequence or execution order as described in the timing diagram. Communication between objects is described using sequence and communication diagrams. Sequence diagram focuses on the times that messages are sent. Communication diagram focuses on object roles. A Communication model can be used to show use case objects and the sequence of messages passed between them.

- *Definition* 1: Proposed OOCPN structure:

The proposed OOCPN structure is defined by the tuple n = ($\sum$, *Pg*, P, Fp, T, SubT, A, N, C, G, E, M_0, R) where:

$\sum$: is a finite set of non-empty types, colour sets

Pg: $\{pg_0,.,pg_n\}$ is a set of pages, where pg_0 is the main page

P: $\{p_1,p_2, ...,p_n\}$ is a finite set of places

Fp: $\{fp_1, fp_2, ...,fp_n\}$ is a finite set of fusion places

T: $\{t_1, t_2, ..., t_n\}$ is a finite set of transitions

SubT = $\{Subt_1, ..., Subt_n\}$ is a finite set of substitution transitions

A: $A \subseteq T \times P \cup P \times T$ represents a set of directed arcs

N: $A \rightarrow T \times P \cup P \times T$ is a node function

C: $P \rightarrow \sum$ is a colour function

G: is a guard function

E: is an arc expression function

M_0: $P \rightarrow C$ is the initial (coloured) marking

R: $\{r_1, ...,r_n\}$ is a finite set of consistency and integrity rules

- *Definition* 2: OOCPNs model Relations and Rules:

The Proposed transformation rules are used to transform the UML diagrams' elements to OOCPN elements. OOCPN elements are grouped together according to the UML diagrams' relations as follows: Let *O* be an OO software system represented by a set of UML diagrams elements (E_o) where $E_o = \{E_1, E_{2,.......}, E_n\}$. Let $TR_o = \{TR_1, TR_{2,...}TR_n\}$ be the set of transformation rules. Let $OOCPN_o= \{OOCPN_1, OOCPN_{2,...} OOCPN_n\}$ be the set of equivalent OOCPN elements of E_o. The transformation rule can defined between $\{E_j, OOCPN_j\}$ as follows: $\forall$ Diagram element $\in E_o$: $E_j \underset{TRj}{\Longrightarrow} OOCPN_j$ *//j is a diagram element*. The OODs are organized in OOCPNs as a set of {S, B, and I}. *Where S: are the UML Structural diagrams' elements, B: are the UML Behavioural diagrams' elements, and I are the UML Interaction diagrams' elements*. The OODs elements in the OOCPNs are a set of: $\{S (E_1,..,E_n), B (E_1, ..,E_n), I(E_1,.., E_n)\}$. S elements are a set of: $\{CD(E_1,..,E_n), OD (E_1,..,E_n), PD(E_1,..,E_n), CoD(E_1,..,E_n), DD(E_1,..,E_n), CSD(E_1,..,E_n)\}$. B elements are a set of: $\{UCD (E_1,..,E_n), AD (E_1,..,E_n), SCD(E_1,..,E_n)\}$. I elements are a set of: $\{SD (E_1,..,E_n), CommD (E_1,..,E_n), TD (E_1,..,E_n), IOD (E_1,.., E_n\}$. *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*. The proposed transformation rules include information about the following: Rules to transform UML diagrams elements into OOCPN as discussed in Definition 2. *Consistency and integrity rule(s)* to maintain consistency and integrity during the transformation and after updating the OOCPN model components. These rules have the structure: If (set of input conditions) Then (set of output conditions) Else (set of output conditions). Consistency Rules are listed in section 4. UML to proposed OOCPN structure transformation rules are available online on[1].

[1] https://docs.google.com/open?id=0BxDI0GvG3aitNF9NeGF1WFZxb1E

3.2. Consistency Rule Design

In the proposed framework, the UML structural, behavioural, or interaction diagram elements are all subject to change to accommodate new requirements. The scope of a change is determined by its impact (local or global). Each update operation is represented as a template. Description for each template is available online on[1]. New changes are represented as rules to update diagram elements or relations incrementally based on diagrams relations; if a change to an element is based on other elements, those elements must exists. Consistency and integrity rules to maintain the consistency between UML diagrams and their relations are proposed in section 4. These rules are checked and applied during the change impact and traceability analysis process. Rule conditions, actions, and pre and post conditions are also considered. All consistency constraints are maintained before and after the new changes have been updated. If any one of these constraints is not satisfied then it is rejected in accordance with Rules 1 to 3 as formulated in Section 4. Data integrity is a critical issue and needs to be validated against certain constraints before and after applying a change. Integrity rules express constraints and define the acceptable relationships between data elements, as well as ensuring completeness. In this research, these rules are checked incrementally after each update operation, and any sequence of updates that occurs must not result in a state that violates any of the constraints. For example, the proposed rules disallow the deletion of referenced data.

4. Components Affected by the Change

In the proposed framework, determining the UML diagram elements affected by a change are determined based on the object dependency graph of the diagram objects and their relations. Control flow dependency and other dependencies such as inheritance, aggregation, encapsulation, polymorphism, and dynamic binding are supported. Figure 3 shows a graph the dependency between the UML diagrams. Any update operation in a structural diagram will cause a change in the behavioural and interaction diagrams. Also the behavioural and interaction diagrams are interdependent; if a change has happened in one of the behavioural diagrams, then, it will affect at least one interaction diagram and vice versa. The following formal definitions (Definitions 1 to 3) are used to determine the dependencies between the UML diagram elements.

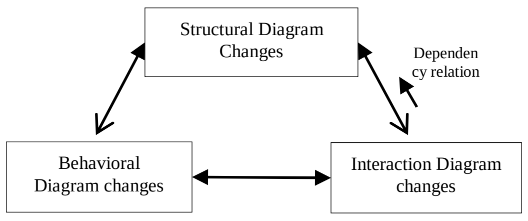

Structural Diagram Changes
Dependency relation
Behavioral Diagram changes
Interaction Diagram changes

Figure 3. UML Diagrams dependency graph.

- *Definition* 3: Impact related elements:

Let X, Y ∈ E, where E is the set of UML diagram elements where X ≠ Y, Y is said to be an impact related element of X, if Y is changed then X is considered changed (Briand, Labiche, & O'Sullivan, 2003). In the proposed framework, this definition can be used to determine the impact of change between any Structural diagram's elements (S), Behavioural diagram's elements (B), and Interaction diagram's elements (I) according to the following relations:
∀ X∈ S, Y ∈ B, Z ∈ I: X is an impact-related element of Y and Z, If (X is updated) Then (Y and Z are changed elements); ∀ X∈ S, Y ∈ B, Z ∈ I: Y is an impact-related element of X and Z, If (Y is updated) Then (X and Z are changed elements); ∀ X∈ S, Y ∈ B, Z ∈ I: Z is an impact-related element of X and Y, If (Z is updated) Then (X and Y are changed elements).

- *Definition* 4: Reflexive relation: D: is the Change Impact (CI) dependency, A: is a UML diagram. The reflexive relation is defined by(Lee, 1998):

A D A: *A depends on itself. if A is impacted, it will impact itself.* This definition describes vertical consistency type. The reflexive relations are: S D S, B D B, and I D I

- *Definition* 5: Transitive relation:

Suppose X, Y, and Z are UML diagrams, then, the transitive relation is defined by (Lee, 1998):

X D Y and Y D Z ⇒ X D Z *This means that if X impacts Y and Y impacts Z, then X impacts Z.* In the proposed framework, examples of the transitive relations between S, B and I are: S D B and B D I ⇒ S D I , S D I and I D B ⇒ S D B. For example, a change to the class diagram will affect the activity diagram (direct impact), and a change to the activity diagram will affect the interaction overview diagram (direct impact). As a result, a change to the class diagram will affect the interaction overview diagram (indirect impact). The change impact dependencies between the UML structural, behavioural, and interaction diagrams are defined using the relations between diagrams. The UML diagram relations are used to determine and classify all types of change in UML diagrams, and the impact on other diagram elements. Horizontal, vertical, and evolutionary traceability and consistency types are supported to maintain consistency and compatibility between the UML diagrams and their versions. The horizontal relation between the diagram elements is affected by a change and the change types can be described formally as in Definition 6. The evolutionary relation between the diagram versions can be described formally as in Definition 7. The change impact is determined for both direct and indirect change effects. A direct effect occurs when the change to one diagram element directly impacts the definition of another diagram element. An indirect effect occurs when the impacted diagram element in turn impacts other diagram elements.

- *Definition* 6: Relation between UML diagram elements and change types:

Let *O* be an OO software system represented by a set of UML diagram elements (E_o) where $Eo = \{E_1, E_{2,\ldots\ldots,} E_n\}$. Let $To = \{t_1, t_{2,\ldots\ldots,}\ t_n\}$ be the set of change types that can be carried out on *O* such that for a given change$\{t_j, E_j\}$, we can define: $F_{impact} \{t_j, E_j\} \rightarrow \{E_1, E_{2,\ldots,} E_k\}$(AJILA, 1995). *where k is the number of the effected diagram elements.* Where F_{impact} is the impact function, $\{E_1, E_{2,\ldots,}\ E_k\}$ is the set of diagram elements effected by applying change (t_j) on element (E_j). The F_{impact} can be extended to include the UML diagrams Categories (C): S, B, and I as in the following: $F_{impact} \{t_j, C_j\} \longrightarrow \{S\ (E_1,..,E_k), B\ (E_1, E_{2,\ldots,}\ E_k), I(E_1,..,E_k)\}$. F_{impact} for the structural diagrams: $F_{impact} \{t_j, S_j\} \rightarrow \{CD\ (E_1,.., E_k), OD\ (E_1,.., E_k), PD(E_1,.., E_k), CoD(E_1,.., E_k), DD(E_1,.., E_k), CSD(E_1,.., E_k)\}$.

F_{impact} for the behavioural diagrams: $F_{impact} \rightarrow \{t_j, B_j\}$ $\{UCD(E_1,.., E_k), AD\ (E_1,.., E_k), SCD(E_1,.., E_k)\}$

F_{impact} for the interaction diagrams: $F_{impact} \rightarrow \{t_j, I_j\}$ $\{SD(E_1,.., E_k), CommD\ (E_1,.., E_k), TD\ (E_1,.., E_k), IOD\ (E_1,.., E_k\}$. This definition describes horizontal consistency type.

- *Definition* 7: Relation between UML diagrams' versions: Based on the definition of Fimpact, the new version created from the impacted diagram elements is

  $F_{impact} \{t'_j, E'_j\} \longrightarrow \{E'_1, E'_{2,\ldots,}\ E'_{k.}\}$.

The new version from the UML diagrams Categories (C'): S', B', and I' are: $\{S'\ (E'_1, E'_{2,\ldots,}\ E'_k), B'\ (E'_1, E'_{2,\ldots,}\ E'_k), I'(E'_1, E'_{2,\ldots,}\ E'_k)\}$. Such that: $\forall$ E_j E_o, If (E_j is changed) Then (E'_j is created as new version from E_j). The new version of the structural diagrams is: $\{CD'(E'_1,.., E'_k), OD'\ (E'_1,.., E'_k), PD'(E'_1,.., E'_k), CoD'(E'_1,.., E'_k), DD'(E'_1,.., E'_k), CSD'(E'_1,.., E'_k)\}$. The new version of the behavioural diagrams is: $\{UCD'(E'_1,.., E'_k), AD'\ (E'_1,.., E'_k), SCD'(E'_1,.., E'_k)\}$. The new version of the interaction diagrams is: $\{SD'(E_1,.., E'_k), CommD'\ (E'_1,.., E'_k), TD'\ (E'_1,.., E'_k), IOD'\ (E'_1,.., E'_k\}$

This definition describes the relations between the UML diagram versions, and the evolutionary consistency types. Definitions 1 to 5 are considered as change impact and dependency rules. The dependency between a business model's components and its impact analysis can be supported efficiently through the proposed change impact and traceability templates which include the following information for each change in the UML diagram elements: *The Change Type* represents the rule. It could be creating, deleting, or modifying a diagram element. *The Change Impact* value is "LC" for the local change, "GC" if the change affects other diagrams' elements or "Null" if the update operation is not allowed. *The Affected Diagrams (Dependency)* is the list of the affected diagrams. *Consistency and Integrity Rules* are designed to maintain the consistency between UML diagrams and their relations. These rules will be checked and applied during the change impact and traceability analysis process. The structure of the rules is provided in Section 3.2. This research proposes the following general consistency and integrity rules:

- *Rule* 1: Deleting/Modifying a referenced element

If (an update is to delete/modify a referenced element) Then (Deleting/Modifying the referenced element is not allowed) *// a referenced element is an element defined by another diagram. For example, diagrams' attributes are defined by the CD.*

The change impact value will be "Null", and the dependency value is "None". The change impact and dependency value for following update operations are determined based on Rule 1:

a) Deleting the following diagram elements: A CD attribute, operation, class, class inheritance, association, or navigability arrow, an object in the OD, SD, or CommD, a UCD actor or use case, and a SCD state or event.
b) Modifying the following diagram elements: A CD attribute name, operation name, class name, inherited class name, navigability arrow direction, polymorphic operation name, or interface element name, an object name in the OD, SD, or CommD, a PD package class name, a CoD & DD component operation name, a CSD part/port name, a UCD actor or use case name, a SCD state or event name, a SD message name or a message attribute name, an IOD activity or interaction diagram element name, and TD task name.

- *Rule* 2: Creating/Deleting/Modifying a non-referenced element

If (an update is to delete a non-referenced element) Then (The change impact is local) *A change impact value will be "LC", and the dependency value is "None".* The change impact and dependency value for following update operations are determined based on Rule 2:

a) Creating the following diagram elements: A CD Value, an OD instance variable or variable/message data type, a SD note, and an order of priorities in the TD.
b) Deleting the following diagram elements: A CD multiplicity range, interface, polymorphic operation, or role name, a SD message, an IOD activity or interaction diagram element, a TD task, an OD instance variable, a SD note, and an order of

priorities in the TD.

c) Modifying the following diagram elements: A PD package name, a CoD & DD node name, an AD start or end node name, a SCD initial or final state, an OD instance variable or variable/message data type, a SD note, and an order of priorities in the TD.

- *Rule* 3: Consistency and Integrity constraints
  - *Rule* 3.1: The class attribute name and the association role name cannot have the same name [3].
  - *Rule* 3.2: Two associations with the same name and role name are not allowed
  - *Rule* 3.3: No private/protected attribute or operation can be accessed by an operation of another class
  - *Rule* 3.4: All diagram attributes/operations must be defined in the CD
  - *Rule* 3.5: A cycle is not allowed in any directed path of aggregation links
  - *Rule* 3.6: For any update operation, the affected diagrams should also be updated
  - *Rule* 3.7*:* A diagram element cannot update an attribute if the attribute changeability is not "changeable"

Change impact and traceability analysis templates are proposed for the changes in UML diagram elements. These templates can be applied to detect the direct or indirect change effect for all diagrams' elements. These templates also describe the change impact and traceability analysis information for UML diagrams' elements. This information is used in the vertical and horizontal consistency types between UML diagrams. Algorithm 1 given below is used to find the diagram elements' affected by the change based on the objects dependency graph. Data dependency is checked as pre and post conditions for each change.

Algorithm 1: Components Affected by the Change

```
Input: Diagram Name (N), Diagram Elements, Change Impact (CI). Output: Diagrams Affected (Dependency).
Process: O: an OO software system represented by a set of UML diagrams elements ($E_o$). D: CI dependency. $N_o$: set of UML diagrams' elements. $N_j$: specific element in the diagram. S: Structural diagrams' elements, B: Behavioural diagrams' elements, I: Interaction diagrams' elements
  Begin If (CI is LC) Then
      - $N_j$ D $N_j$ // $N_j$ depends on itself. It means that if $N_j$ is impacted, it will impact itself.
      - $\forall N_j \in N_o$, If ($N_j$ is changed) Then ($N'_j$ is created as new version from $N_j$)
  Else //global changes If ($N_j$   S) Then
      - $\forall X \in S$, $Y \in B$, and $Z \in I$: X is an impact related element of Y and Z,
        If (X is updated) Then (Y, Z is a changed element)
      - X', Y', and Z' are created as new versions from X, Y, and Z respectively.
  Else If ($N_j$   B) Then
      - $\forall X \in S$, $Y \in B$, $Z \in I$: X : Y is an impact related element of X and Z, If (Y is updated) Then (X and Z is a changed element)
      - X', Y', and Z' are created as new versions from X, Y, and Z respectively.
  Else (If $N_j$   I) Then
      - $\forall X \in S$, $Y \in B$, $Z \in I$: X : Z is an impact related element of X and Y,
      - If (Z is updated) Then (Y is a changed element)
      - X', Y', and Z' are created as new versions from X, Y, and Z respectively.
  endif endif endif Versions update endif End
```

5. Proposed Change Management Framework Modeling , Simulation, and Analysis

Change is inevitable in the software product lifecycle. Change impact and traceability analysis are important activities in software maintenance process to analyze the possible effects of software changes. A software design is often modeled using a collection of UML diagrams. The use of UML diagrams in modeling large-scale systems leads to a large number of interdependent diagrams. It is necessary to preserve the diagrams consistency, since they are updated continuously. To accommodate the change in the software process, a change management framework has been proposed to determine the change effect in the UML diagrams' elements. The proposed framework can be applied to detect the elements affected by the change in the systems design modeled using UML diagrams. 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, and the relationships between the model diagrams. In this section, we will discuss the modeling and simulation and analyzing the proposed change management framework. CPN Tools version 3.4 [26] is used to model and simulate the proposed OOCPN structure and to validate the proposed change management framework. In comparing with other approached that discussed in the related works, the proposed framework discussed and provided the transformation rules between UML diagram from different perspectives using UML structural, behavioural, and interaction diagrams rather than a sequence of activities. This transformation included the consistency rules bases on the diagrams relations. Algorithm 1 has been proposed to determine the change impact and the dependency between the diagrams' elements. Corrective and evolutionary changes are supported. Figure 4. Shows the hierarchy for the change levels in the proposed framework. The change levels are used to determine the distances between the changed element and the impacted elements. The change distance is calculated according to the following rule: If (the change in S, B, or I is local) Then (change distance is 1) Else (change distance is 2). *// the number* *of affected diagrams (n) by the change is n ≥ 1 to be explained in.* Change impact and traceability analysis templates have been proposed to determine and classify types of changes in UML diagrams and their impact on other diagrams.

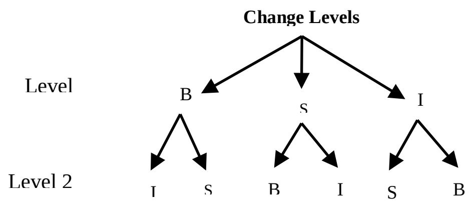


Figure 4. Change levels (traceability distance).

The diagrams consistency is checked according to the consistency and integrity rules provided in each template. This includes the vertical, horizontal and evolutionary consistency types. The dependency between UML diagrams has also been defined formally as discussed in the Definitions 1 to 5. The proposed framework in comparing with the state of art proposed the incremental checking of consistency between diagrams; we applied the same idea of incremental syntax checking in CPN Tools.

- ∃ *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, behavioural, 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)

The proposed change management framework trace the dependency and to determine the effect of change of UML diagrams elements incrementally; it is used to check the consistency, impact, and traceability after creating, deleting, or modifying any diagram element by applying the same idea of syntax checking incrementally in CPNs. also a comparison between two versions from the same diagram is supported. Figure 5 shows a complete example for modeling a change type: adding a new CD operation and its consistency rule this include the change history and version (File Update substitution transition). In all steps, initial token are provided for all nodes in order to trace the simulation process through transferring these tokens from the input to output places. In CPN-Tool, all the CPN models can be translated to Java code using the "Export to Java code" provided in the Net tool box. Figure 7 summarizes the distribution of the proposed templates and the transformation rules on the UML diagrams categories. 22 templates are proposed for the structural diagrams, 18 templates are proposed for the behavioral diagrams, and 13 templates are proposed for the interaction diagrams. Some of these templates are shared between some diagrams based on diagrams relations. For example, the same template is proposed for the activity diagram and sequence diagram iteration /loop changes. Information about the number of affected diagrams by updating each UML diagram and the number of update operation supported is summarized in Figure 8. Self, direct, and indirect dependencies are considered. The proposed framework can be implemented for actual deployments in any system modelled using OO diagrams, such as those in large universities, industrial factories, large or small companies, social networking systems to provide software model analysis and design. UML diagram is used to model the social network systems, dealing with changes in UML diagrams will also help in enhancing the social media software's change management. Figure 6 summarize the number of transformation rules proposed for each diagram.

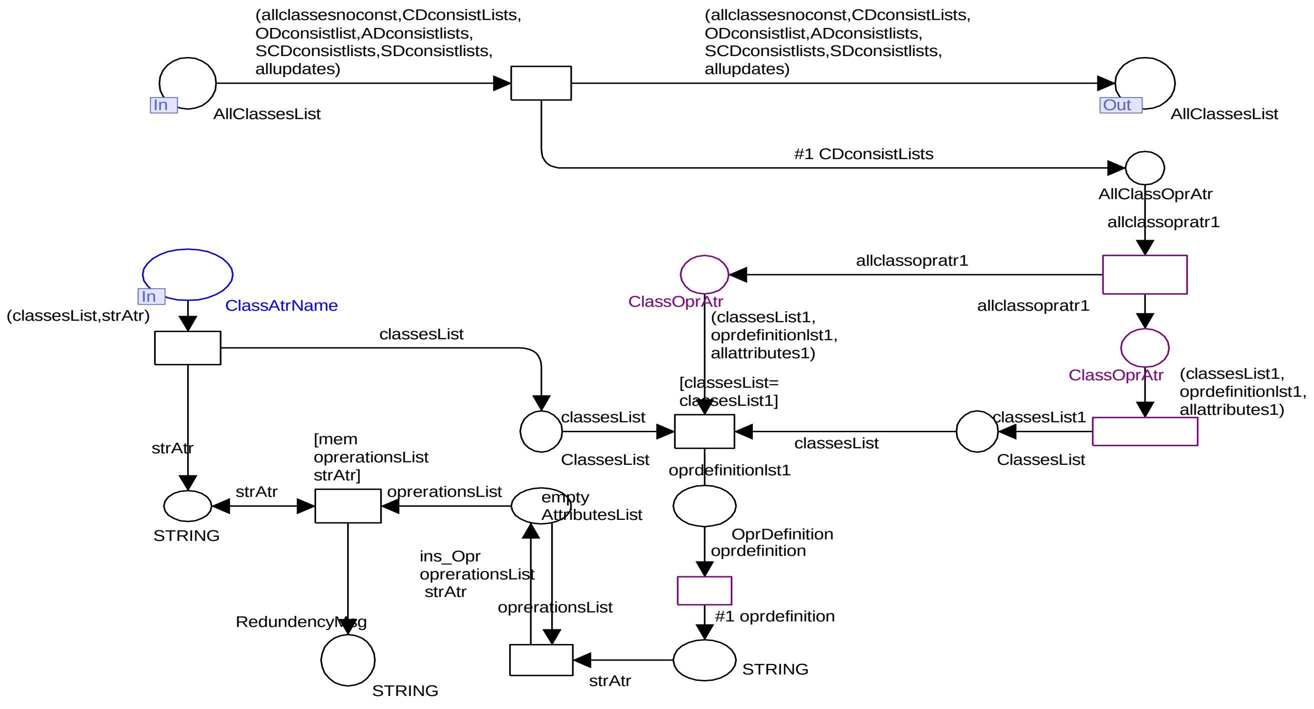

(allclassesnoconst,CDconsistLists, ODconsistlist,ADconsistlists, SCDconsistlists,SDconsistlists, allupdates)
(allclassesnoconst,CDconsistLists, ODconsistlist,ADconsistlists, SCDconsistlists,SDconsistlists, allupdates)
In
AllClassesList
Out
AllClassesList
#1 CDconsistLists
AllClassOprAtr
allclassopratr1
allclassopratr1
allclassopratr1
In
ClassAtrName
(classesList,strAtr)
ClassOprAtr
(classesList1, oprdefinitionlst1, allattributes1)
ClassOprAtr
(classesList1, oprdefinitionlst1, allattributes1)
classesList
[classesList= classesList1]
classesList
ClassesList
classesList1
classesList
ClassesList
strAtr
[mem opretationsList strAtr]
strAtr
opretationsList
empty AttributesList
oprdefinitionlst1
STRING
OprDefinition
oprdefinition
ins_Opr opretationsList strAtr
RedundencyMsg
opretationsList
#1 oprdefinition
STRING
strAtr
STRING

Figure 5. Adding new class diagram operation.

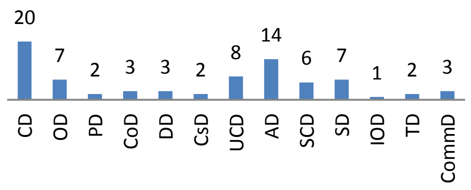
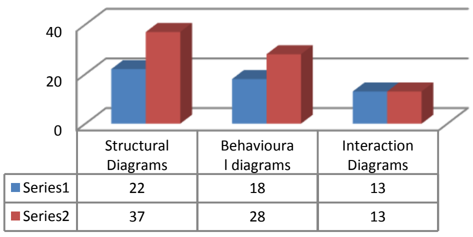
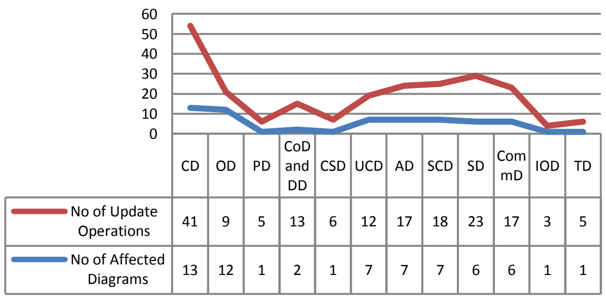


Figure 6. Number of transformation rules for each diagram.



	Structural Diagrams	Behavioural diagrams	Interaction Diagrams
🟦 Series1	22	18	13
🟥 Series2	37	28	13

Figure 7. Number of proposed templates (Series 1) and transformation rules (Series 2) for each diagrams category.



	CD	OD	PD	CoD and DD	CSD	UCD	AD	SCD	SD	Com mD	IOD	TD
No of Update Operations	41	9	5	13	6	12	17	18	23	17	3	5
No of Affected Diagrams	13	12	1	2	1	7	7	7	6	6	1	1

Figure 8. Number of update operations supported and number of diagrams affected by updating each UML diagrams.

6. Conclusions

As software evolves, analysis and design models need be modified, accordingly. To cope with changes in the software process, in this research, a novel approach for a change management framework was proposed to manipulate the change effect in the UML diagrams' elements. In this framework, UML diagrams are modelled from different perspectives using UML structural, behavioural, and interaction diagrams. The proposed framework can be applied to detect the diagram elements affected by a change in a system design modelled using UML diagrams by utilizing the proposed templates. This framework can be used to control the change of UML diagrams by identifying and managing the model changes, ensuring the correctness and consistency of the models, identifying the impact of changes based on the relationships between diagrams, and analyzing the performance. The proposed templates determine and classify the types of changes in UML diagrams and their impact on other diagrams. The consistency between diagrams is checked according to the consistency and integrity rules provided in each template. This includes the vertical, horizontal, and evolutionary consistency types. An algorithm has been proposed to find out all the possibly affected elements if these changes happened based on the proposed consistency and integrity rules. CPNs Tools toolboxes are used to model and simulate the proposed framework. The future work of this research is to develop a change management patterns for all templates and transformation rules provided in this research.

References

[1] Abma B., Evaluation of Requirements Management Tools With Support for Traceability-Based Change Impact Analysis, Master's Thesis,

University of Twente, 2009.

[2] Bolloju N., Schneider C., and Sugumaran V., "A Knowledge-Based System for Improving The Consistency Between Object Models and Use Case Narratives," *Expert Systems with Applications*, vol. 39, no. 10, pp. 9398-9410, 2012.

[3] Briand L., Labiche Y., and O'sullivan L., "Impact Analysis and Change Management of UML Models," *in Proceedings of International Conference on Software Maintenance*, Amsterdam, pp. 256-265, 2003.

[4] Briand L., Labiche Y., and Yue T., "Automated Traceability Analysis for UML Model Refinements," *Information and Software Technology*, vol. 51, no. 2, pp. 512-527, 2009.

[5] Chen C., She C., and Tang J., "An Object-Based, Attribute-Oriented Approach for Software Change Impact Analysis," *in Proceedings of IEEE International Conference on Industrial Engineering and Engineering Management*, Singapore, pp. 577-581, 2007.

[6] Dang D. and Gogolla M., "An OCL-Based Framework for Model Transformations," *Journal of Science Computer Science and Communication Engineering*, vol. 32, no. 1, 2016.

[7] De Lucia A., Fasano F., and Oliveto R., "Traceability Management for Impact Analysis," *in Proceedings of Frontiers of Software Maintenance*, Beijing, pp. 21-30, 2008.

[8] Egyed A., "Instant Consistency Checking For The UML," *in Proceedings of the 28th International Conference on Software Engineering*, 2006.

[9] Egyed A., "Fixing Inconsistencies In UML Design Models," *in Proceedings of 29th International Conference on Software Engineering*, Minneapolis, pp. 292-301, 2007.

[10] Egyed A., "Uml/Analyzer: A Tool for The Instant Consistency Checking of Uml Models," *in Proceedings of 29th International Conference on Software Engineering*, Minneapolis, pp. 793-796, 2007.

[11] Egyed A., "Automatically Detecting and Tracking Inconsistencies in Software Design Models," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 188-204, 2011.

[12] Kchaou D., Bouassida N., and Ben-Abdallah H., "Managing The Impact of UML Design Changes On Their Consistency and Quality," *Arabian Journal for Science and Engineering*, vol. 41, no. 8, pp. 2863-2881, 2016.

[13] Khalil A. and Dingel J., "Supporting the Evolution of UML Models in Model Driven Software Development: A Survey," Technical Report, 2013.

[14] Lehnert S., "A Review of Software Change Impact Analysis," Technical Report, Ilmenau University of Technology, 2011.

[15] Lucas F., Molina F., and Toval A., "A Systematic Review of UML Model Consistency Management," *Information and Software Technology*, vol. 51, no. 12, pp. 1631-1645, 2009.

[16] Mahmood Z. and Mahmood R., "Category, Strategy and Validation of Software Change Impact Analysis," *International Journal of Engineering and Computer Science*, vol. 4, no. 4, pp. 11126-11128, 2015.

[17] Mens T., Van Der Straeten, R., and Simmonds, J., "A Framework for Managing Consistency of Evolving UML Models," *Software Evolution with UML and XML*, pp. 1-31, 2005.

[18] Mohan K., Xu P., Cao L., and Ramesh B., "Improving Change Management in Software Development: Integrating Traceability and Software Configuration Management," *Decision Support Systems*, vol. 45, no. 4, pp. 922-936, 2008.

[19] Niepostyn, S., "The Sufficient Criteria for Consistent Modelling of the Use Case Realization Diagrams with a New Functional-Structure-Behaviour Uml Diagram," *Przegląd Elektrotechniczny Sigma NOT*, vol. 2, pp. 31-35, 2015.

[20] Puissant J., Resolving Inconsistencies in Model-Driven Engineering using Automated Planning, PhD Thesis, Universit de Mons, 2012.

[21] Rajabi B. and Lee S., "A Study of The Software Tools Capabilities in Translating UML Models to PN Models," *International Journal of Intelligent Information Technology Application*, vol. 2, no. 5, pp. 224-228, 2009.

[22] Rajabi B. and Lee S., "Consistent Integration Between Object Oriented and Coloured Petri Nets Models," *The International Arab Journal of Information Technology*, vol. 11, no. 4, pp. 406-415, 2014.

[23] Sapna P. and Mohanty H., "Ensuring Consistency in Relational Repository of UML Models," *in Proceedings of 10th International Conference on Information Technology*, Orissa, pp. 217-222, 2007.

[24] Sharaff A., "A Methodology for Validation of OCL Constraints Using Coloured Petri Nets," *International Journal of Scientific and Engineering Research*, vol. 4, no. 1, 2013.

[25] Shinkawa Y., "Inter-Model Consistency in Uml Based on Cpn Formalism," *in Proceedings of 13th Asia Pacific Software Engineering Conference*, Kanpur, pp. 411-418, 2006.

[26] Westergaard M. and Verbeek H., *CPN Tools*. Available from: http://cpntools.org/ Last Visited, 2013.

Bassam Rajabi received his PhD degree in Software Engineering from 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 to 2017 with Wajdi University College of Technology-Palestine. Currently, he is the dean of Ibrahimieh Community College. His areas of interest are Software Design and Modeling Techniques.

Sai Peck Lee is a professor at the Department of Software Engineering, University of Malaya. She obtained her Ph.D. degree in Computer Science from Université Paris 1 Panthéon-Sorbonne. Her current research interests include Object-Oriented Techniques and CASE tools, Software Reuse, Requirements Engineering and software quality. She is a member of IEEE and currently in several experts review panels, both locally and internationally.