Volume: 08 Issue: 05 Pages: 3188-3200 (2017) ISSN: 0975-0290

Use Case Modeling in Software Development: A Survey and Taxonomy

Zahra Rashidi
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
zrashi@ce.sharif.edu
Zeynab Rashidi
Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran
zeynabrashidi@aut.ac.ir
Hassan Rashidi
Department of Mathematics and Computer Science, Allameh Tabataba'i University, Tehran, Iran
hrashi@atu.ac.ir

----------**ABSTRACT**----------

Identifying use cases is one of the most important steps in the software requirement analysis. This paper makes a literature review over use cases and then presents six taxonomies for them. The first taxonomy is based on the level of functionality of a system in a domain. The second taxonomy is based on primacy of functionality and the third one relies on essentialness of functionality of the system. The fourth taxonomy is concerned with supporting of functionality. The fifth taxonomy is based on the boundary of functionality and the sixth one is related to generalization/specialization relation. Then the use cases are evaluated in a case study in a control command police system. Several guidelines are recommended for developing use cases and their refinement, based on some practical experience obtained from the evaluation.

Keywords -**Use cases, Taxonomy, Software Engineering.**

Date of Submission: April 10, 2017 Date of Acceptance: April 18, 2017

I. INTRODUCTION

Domain analysis paves the way from gathering requirements to an object-oriented analysis and modeling. In turn, system analysis feeds back into domain analysis by demanding richer and more refined definitions for concepts and scopes. Fig. 1 shows that domain analysis feeds conceptual modeling, but is also updated and refined through that modeling[1].

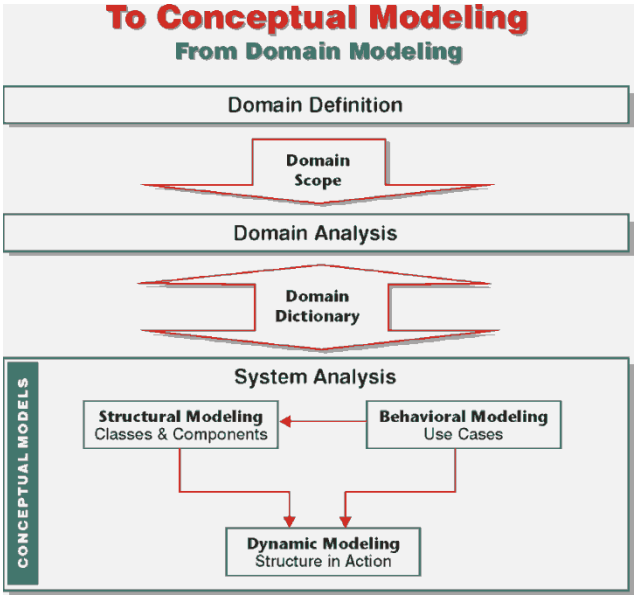

Figure 1- Steps in domain analysis

Domain analysis discovers and defines business concepts within the context of the problem space. Use case modeling channels, transforms and expands these concepts into a model of system behavior (see [2], [3], [4], [5], [6]).

Use cases are the first step towards conceptual modeling in which a set of related use cases provide the behavioral model of a system (see Fig. 1). The boundaries of the system or the subsystem depicted by use cases are defined by domain definition. The starting point of use cases are the concepts discovered through domain analysis—primarily, but not limited to, those categorized as processes. Use case modeling is a set of use cases that, together, describe the behavior of a system. A use case is a unit of this model in which is a description of an interaction that achieves a useful goal for an actor.

A use case can be a textual narrative, but it must have four well-defined components to qualify as a use case. The first component is a goal as the successful outcome of the use case and second one are stakeholders whose interests are affected by the outcome (including actor(s) who interact with the system to achieve the goal. The third component is a system that provides the required services for the actors, and forth one is a step-by-step scenario that guides both the actor(s) and the system towards the finish line. The narrative of a use case is made up of one or more flows. The normal flow is the best-case scenario that results in the successful completion of the use case. An alternate flow exists only if conditional steps are needed. It may have sub-flows if steps in the normal flow contain sub-steps and have exceptions that describe what may

Volume: 08 Issue: 05 Pages: 3188-3200 (2017) ISSN: 0975-0290

prevent the completion of one-step or the entire use case. Table 1 summarizes information associated with a use case and shows a template for developing use cases.

Table 1: A Template for developing a use case and information associated with it

Use Case Name	
Description	This part describes one or two sentence of the use case
Stakeholders/ Actors	This part identifies the actors participating in the use case
Includes	This part identifies the use cases included in it
Extends	This part identifies the use case that it may extend
Pre-Conditions	This part identifies the conditions which must be met to invoke this use case
Details/Flow	This part identifies the details of the use case.
Goal/Post-Conditions	This part identifies the conditions hold at the conclusion of the use case
Exceptions	This part identifies any exceptions that might arise in execution of the use case
Constraints	This part identifies any constraints that might apply
Variants/Alternate Flow	This part identifies any variations that might hold for the use case
Comments	This part provides any additional information, which might be important in the use case

The main motivation of this paper is to survey over the use case modeling and makes six taxonomies for use cases. The structure of remaining sections is as follows. In Section 2, the literature review and taxonomies of use case are presented. In Section 3, a case study for the control command police system is presented. In Section 4, the most important guidelines to develop use cases are recommended. Finally, Section 5 is considered to summary and conclusion.

2. LITERATURE REVIEW AND TAXONOMIES

Use case modeling represents the behavior of a system. A use case details the interaction of entities outside a system, called actors, with the system to achieve a specific goal by following a set of steps called a scenario. Use case modeling is the first step for transforming domain concepts into models for building a solution. A use case is a textual narrative that details how one or more entities called actors interact with a system to achieve a result that is primarily of value to one of them, the primary actor. Various authors define use cases differently:

- Rumbaugh (1994) states that a use case is a description of all of the possible sequences of interactions among the system and one or more actors in response to some initial stimulus by one of the actors [7].
- Iacobson el al (1999) states that a use case specifies a sequence of actions, including a variant that a system performs and that yields an observable result of value to a particular actor [8].
- Cockburn (2000) states that a use case is a collection of possible sequences of interactions between the system under discussion and its external actors, related to a particular goal [9].
- Bruegee and dudoit (2010) state that a use case is initiated by an actor. After its initiation, a use case may interact with other actors, as well. A use case represents a complete flow of events through the system in the sense that it describes a series of related interactions that result from its initiation [10].

The common threads in all of above definitions are actors and sequences of interactions. In this approach, several concepts are important: the goal, the system, and the actor and use case bundle. The goal is the business value to the 'user(s)' of the system who usually initiate the interaction with the system. The system is the application with all of its associated hardware that will be used by the 'users'. An actor is external entity that interacts with a system. A use case bundle is a collection of use cases that are highly correlated with some activity or organizing business element. A use case bundle gives us away to organize our use cases into collections that will help us better understand the functionality of the system that we are developing any large systems.

In the literature, we found several kinds of use cases, including 'High-Level', 'Low-Level', 'Primary' , 'Secondary', 'Essential', 'Concrete', 'Including', 'Extending', 'Starting, 'Stopping' use cases ([11], [12], [13], [14],[15]). We added four other use cases: 'Generalizing', 'Children', 'Frond End' and 'Back End' use cases.

In capturing the functional aspects of the system, one of the difficulties in generating a useful discussion of a system is keeping the description at a consistent level of abstraction. For use cases to be successfully developed, it is necessary to know the dimension of the functional description that one is attempting to capture. Then the analyst can determine the level of detail, primacy of functionality, designing and implementation issues in the information that should be captured. Regarding these issues, we have six taxonomies for the use cases.

2.1 First View: Level Of Functionality

In one dimension, we can distinguish between high-level and low-level functional descriptions of a system:

- **High-LevelUse Case (HLUC)**: It is a black box view of the system by which we deal entirely with the dialog between the actor and the system. High-level use case provides general and brief

Volume: 08 Issue: 05 Pages: 3188-3200 (2017) ISSN: 0975-0290

descriptions of the essence of the business values provided. It is not concerned with how the business values are achieved. For example, managing accounts is a high-level use case in each accounting system of banks.

- **Low-Level Use Case (LLUC)**: It is a white box (or transparent box) view of the system by which we deal with the dialog between the actor and the system and what the system does to provide the functionality. For example, adding an account, updating an account, and deleting an account are low-level use cases that establish detailed activities for managing an account in the account system of banks.

2.2 Second View: Primacy Of Functionality

In a second dimension, we can distinguish between primary and secondary functions of the system.

- **PrimaryUse Case (PRUC)**: These use case provide functions to users that constitute the reason for which the system exists. For example, generating a report is a primary use case in the account system of banks.
- **Secondary Use Case (SEUC)**: These use case are functionality that deals with exceptional and rare cases that may occur in the environment. They allow analysts to capture system behavior under error conditions. These functions are necessary to deliver a robust system. For example, the consistency of database must be controlled in the account system and the crashed data must be recovered.

2.3 Third View: Essentialness Of Functionality

In the third dimension, we can distinguish between the essential and the concrete functions of the system**:**

- **Essential Use Case (ESUC)**: It captures business solutions that are independent of implementation. It is usually depicted as black box models and is independent of hardware and software.
- **Concrete Use Case (COUC)**: It captures business solutions in terms that are design-dependent, like transparent box models. A concrete use case can "extend" an abstract essential use case.

2.4 Forth View: Supporting of Functionality

Some use cases support other use cases. They are called supporting use cases, which are categorized to the following use cases:

- **Including Use Case (INUC):** we must view including use case as a relation that identifies a use case that acts like a subroutine to other use cases. Typically, including use cases will not have actors that initiate them. We can consider these use cases as inheriting actors.
- **Extending Use Case (EXUC):** We can have some situation in which several use cases are identical with the exception of one or two specific subsequences of interactions. In this case, we can extract the common core (base use case) and treat the use cases that differ as extensions of this base use case. Thus, an extend relationship exists between an extension and the core. This allows us to capture in an easy form those situations where the sequences captured in several use cases may differ as the result of a simple conditional at the end of the sequence.

2.5 Fifth View: Boundary of Functionality

In the fifth dimension, we must think about running the system, which concern with start-up and shut down the system.

- **Starting Use Case (SRUC):** These use cases captures the behavior of the system when it is being starting up. They are usually design-dependent, like Transparent Box Models.
- **Stopping Use Case (SPUC):** These use cases capture the behavior of the system when it is being shutting down. They are usually design-dependent, like Transparent Box Models.
- **Frond End Use Case (FEUC):** These use cases capture the behavior of the system when it interacts with users.
- **Back End Use Case (BEUC):** These use cases capture the behavior of the system when it interacts with servers such as Database server and Web server.

These use cases will be the last use cases to be developed because we must wait until sufficient details are known to identify what information must be initialized during startup and preserved during shutdown.

2.6 Six View: Supporting of Inheritance

In the sixth dimension, we consider inheritance between use cases. In this view, there are two kinds of use cases:

- **Generalizing Use Case (GEUC):**If two or more use cases achieve the same goal through different means but share most activities, theGeneralizing Use Case abstracts their common feature into a generalized super-use case (also called the parent).
- **Children Use Case** (CHUC): The children use cases inherit features from the parent, where they override (or specialize) them, or when they add new features.The relationship that exists between the Generalizing use case and the Children is a generalization/specialization relation.

The differences between extending and generalizing use case are subtle, but they are important. The extending use case defines a set of extension points in the basic use case, but the generalization use case does not. With extending use case, the basic core must know that it is going to be extended in order to define the extension points. This is not so with generalizing use case. The extending use case adds to the basic core's functionality, but generalizing use

Volume: 08 Issue: 05 Pages: 3188-3200 (2017) ISSN: 0975-0290

case overridesit so that it totally replaces it, albeit with something similar.

3. PRACTICAL EXPERIMENTS

In order to make specific guidelines for developing use case model, we used a Control Command Police System (CCPS) for which a mini-requirement is briefly described in [16]. This system is extended in [17] and then it is used in our study due to its fertility for reusability in both application and system software. This police service system must respond as quickly as possible to report incidents and its objectives are to ensure that incidents are logged and routed to the most appropriate police vehicle. The most important factors that must be considered which vehicle to choose to an incident include:

- **Type of incident**: some important and worsening events need immediate response. It is recommended that specified categories of response actions are assigned to a definite type of incident.
- **Location of available vehicles**: Generally, the best strategy is to send the closest vehicle to address the incident. Keep in mind that it is not possible to know the exact position of the vehicles and may need to send a message to the car to determine its current location.
- **Type of available vehicles**: some incident need vehicles need and some special incident such as traffic accidents may need ambulance and vehicles with specific equipment.
- **Location of incident**: In some areas, sending only one vehicle for response is enough. In other areas, may be a police vehicle to respond to the same type of accident is enough.
- **Other emergency services such as fireman and ambulance**: the system must automatically alert the needs to these services.
- **Reporting details**: The system should record details of each incident and make them available for any information required.

To identifying major use cases, we must arrive to analyzing domain concepts marked as ‘process’ or ‘function’, but the conversion ratio is not one-to-one. Sometimes we have to break up a process into more than one use case; at other times we might have to combine pieces of multiple processes or functions to arrive at one use case. Other domain concepts, such as objects or business rules might find their way into use cases if the context requires it. Each process that will achieve a useful business goal is definitely one use case. The other fundamentally different processes in which they participate usually result in other use cases. Although the standard practice of documenting use cases is to start from the external event to the system, in certain circumstances, we want to document the use case from the external event to the actor.

At first, we wrote two or three of the most common and simple process. When identifying the use cases, we gave a descriptive name and a one or two sentence description of each. The names of use cases based upon the goal the actor is attempting to achieve and, if necessary, to distinguish variations in the circumstance in which it is invoked. We used a verb to start the name of the use case. Then, we make one or two sentences for description to identify the approximate interaction that is to be captured in the use case. We performed the analysis in an incremental fashion and develop the use case model iteratively. In each iteration, we provided very brief descriptions initially and then refined them so that the goal is to provide more details about the use case.

A couple of experts in IT field helped us to develop the conceptual model of the CCPS. After a long discussion, we prepared Table 2, in which shows the information concerned with specific parts of the template (see Table 1) in different kinds of use cases. It is used to guide the development of use case. The development of high-level, primary, essential use cases requires that analyst identifies the essential business information to establish the business value proposition, the preconditions that must apply for the use case to complete, and the post-conditions that are promised, and any constraints or variations that might exist.

We immediately documented actors identified as the result of describing a use case. These actors needed to be documented in terms of what actions they are required to provide. If multiple actors can initiate the same set of actions, we introduced an abstract actor of which all the others are specializations. A summary of the results in identifying the main ‘high level’ use cases are put into Table 3.

To document use case, we can use ‘use case diagram’ in which there are several basic elements. Use case diagram is a ‘meta-model’-an abstraction of use cases[1]. In its basic form, it displays the boundaries of the system, the name of the use cases, and the actors who interact with each use case. The main use case diagram and activity diagram of this system are depicted in Fig. 2 and Fig. 3, respectively.

Use cases apply to many different paradigms to develop a systems(see[18], [19], [20],[21]). In this study, we apply it to object oriented software development. In this paradigm, after developing use case model we must identify the objects in system and their relationships (See Fig.1 for Structural Modeling). There are many approaches to identify objects ([22], [23], [24], [25], [26], [27], [28],[29]). The Class Diagram of this system is depicted in Fig. 4. In this class diagram, the main classes, are ‘Incident’, ‘Police Staff’, ‘Police Vehicle’, ‘Police Officer’, ‘Director’, ‘Route Manager’, ‘Incident Waiting List’, ‘Response’ and ‘GPS Receiver’. In the figure, the attributes and methods of each class are shown.

Volume: 08 Issue: 05 Pages: 3188-3200 (2017) ISSN: 0975-0290

Table 2: Information concerned with specific parts of the template in different kinds of use cases

Kind of Information	High-Level Primary Essential	Low-Level Primary Secondary Essential	Low-Level Primary Concrete	Low-Level Primary Secondary Concrete
Business Information	Actors	Actors	Actors	Actors
Technological Information		Relations		Relations
Relationships among primary and secondary, extending, and including use cases	Preconditions	Preconditions	Preconditions	Preconditions
Essential information	Post-conditions	Post-conditions	Post-conditions	Post-conditions
High-level information about the interaction between system and actors	Details	Details		
Detailed concrete information that applies to use cases that are not generalizations		Exceptions	Details	Details
Added information included as appropriate for the specific use case	Constraints Variants	Constraints Variants	Constraints Variants	Constraints Variants

Table 3: The main high level uses case identified in the Control Command Police system

Name of the use case	Actors involved	Description
Call Taking	Reporter Of Incident, Police Station Operator, Police Officer, Alarm	An operator in Police Station receives a phone call about an accident at a given location
Incident Registration	Police Station Operator, Police Officer	Police Officer received the call center operator and details of the incident. The system automatically classifies the incident and determines its priorities.
Close Incident	Dispatcher, Record Management System	Dispatcher collects information associated with the accident and the record is saved into the system
Send Report	Primary Police Unit, Dispatcher	A report associated with the incident is collected and send to Dispatcher
Request More Units	Primary unit, Dispatcher	Police Unit requests the new forces, Dispatcher comply with this request or rejects it
Create Response	Dispatcher	Dispatcher makes the response and makes coordination
Unit Management	Police Unit, Dispatcher	Dispatcher handles the incident and monitors the resources/forces allocated to the incident
Send Data	Police Unit, Dispatcher	Incident details (address, number of victims, ...) and call details (field of call, number of units, ...) are sent police unit
Request To incident	Police unit	Police Officers at police cars received information about the accident, then goes to the scene and handles the mission. He/she will be available again for the next mission.
Dispatch Units	Police Unit, Dispatcher	Data, information and how to respond to accident are sent to the selected units
Find Closest Unit	Police Station Operator	Finding the nearest unit to the location of the incident
Get Position of Units	Positioning system	The positioning system finds the closest unit to the scene
Alert Emergency service	Emergency Service	Emergency services such as fire or ambulance system are alerted and called

Volume: 08 Issue: 05 Pages: 3188-3200 (2017) ISSN: 0975-0290

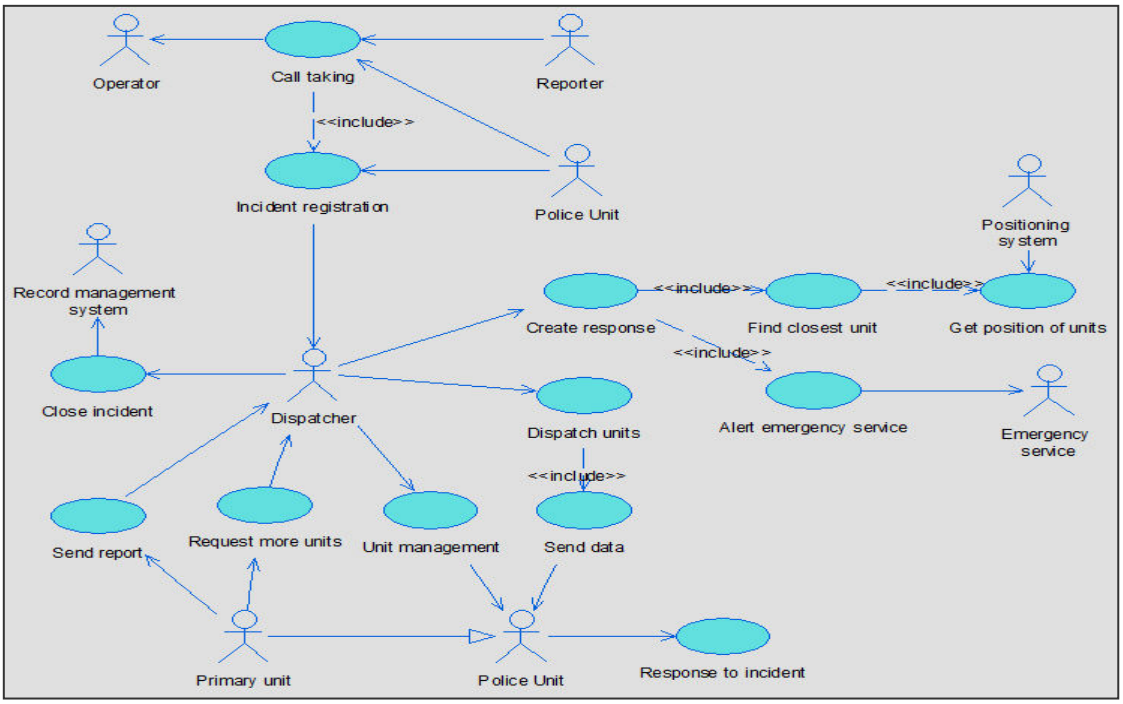

Fig. 2: The main Use Case Diagram of the Control Command Police System

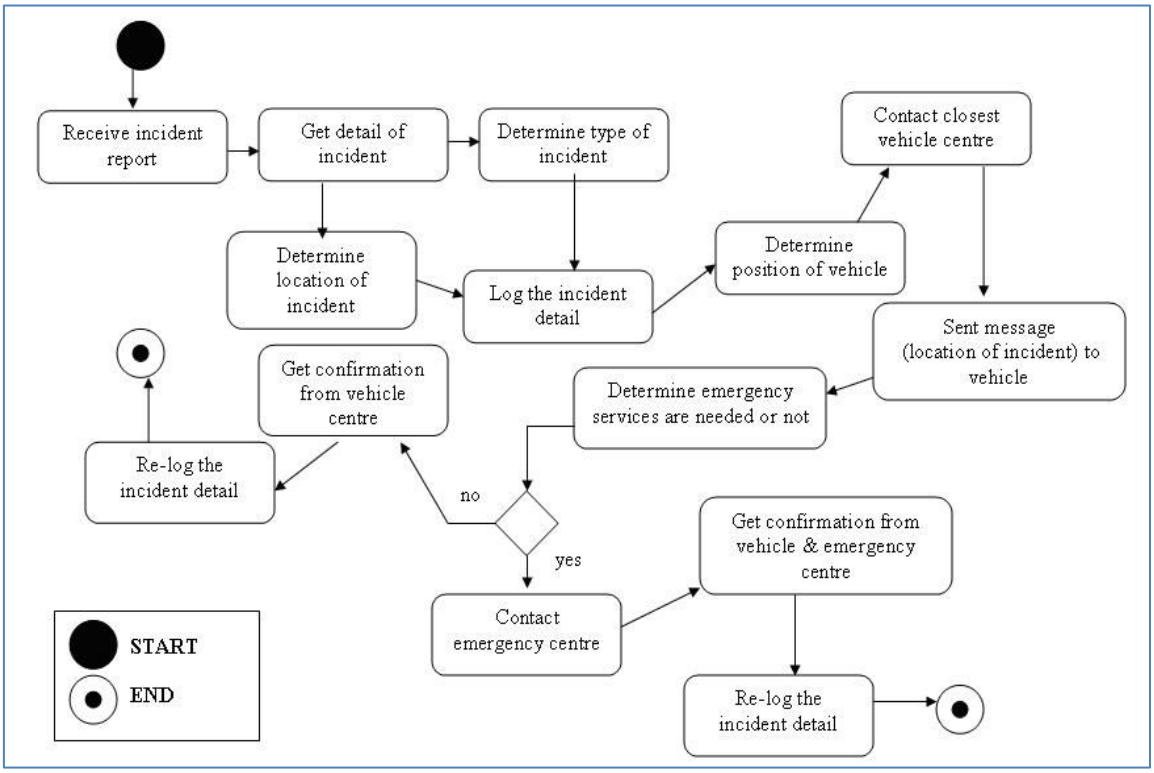

Fig. 3: The Activity Diagram of the Control Command Police System

Use cases and scenarios are not sufficient documentation for conceptual modeling. So we went to Dynamic Modeling step (See Fig.1). In this step, we had to look at ways of documenting the events and documenting the dynamic behavior in a form more suited for programming. In particular, an event list is created and each event in the list must be responded. In the abstraction view, four aspects of an event including 'Label', 'Meaning', 'Source Object' and 'Destination Object' must be specified. We prepared Table 4, as the main event list for the CCPS. Since we did not fully implement this system, the 'Front End and 'Back End' use cases are not in the table.

Volume: 08 Issue: 05 Pages: 3188-3200 (2017) ISSN: 0975-0290

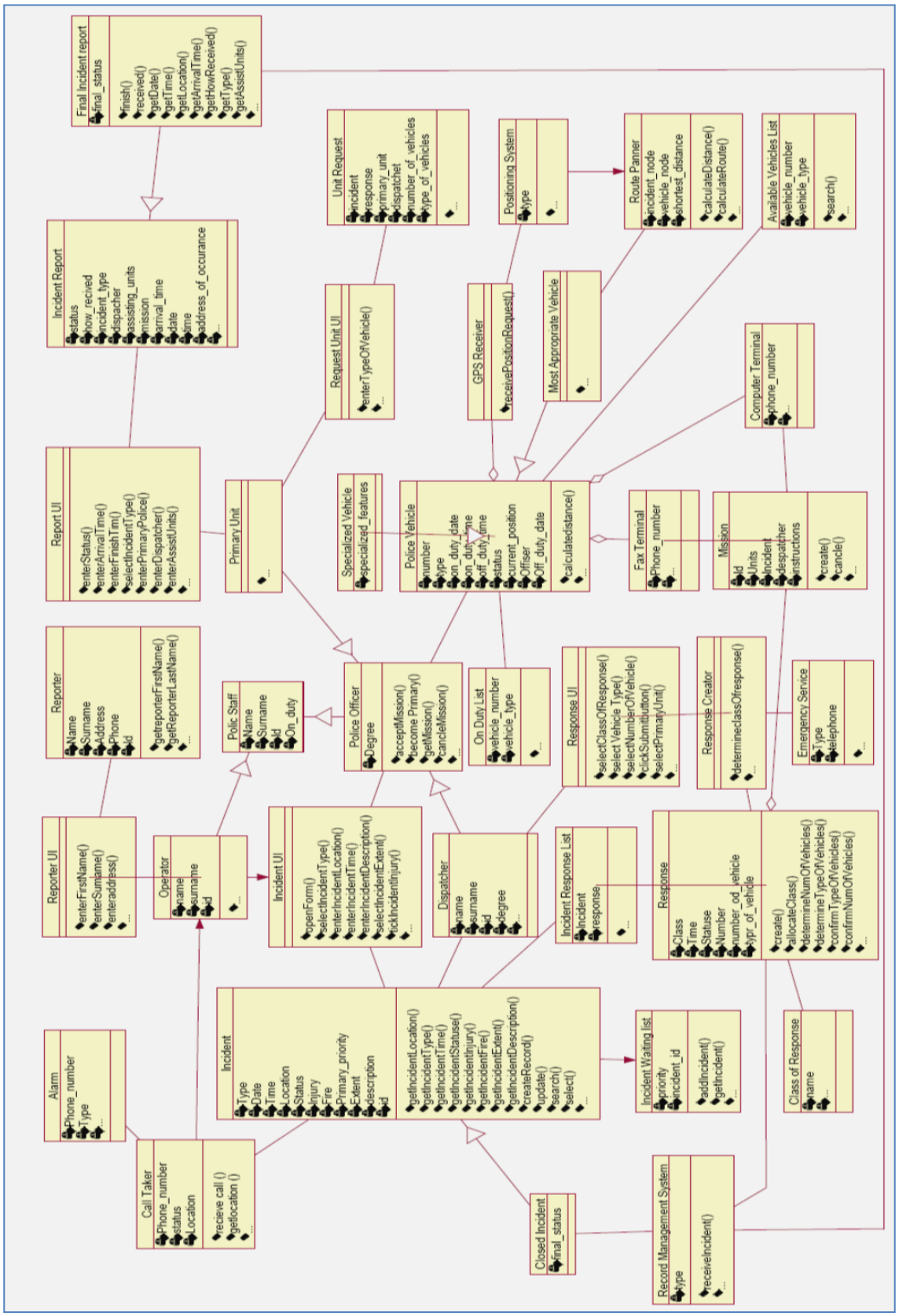

Fig. 4: The Class Diagram of the Control Command Police System

Volume: 08 Issue: 05 Pages: 3188-3200 (2017) ISSN: 0975-0290

Table 4: The main event list captured in the Control Command Police System

	Label	Meaning	Source Object	Destination Object	High Level	Low Level	Primary	Secondary	Essential	Concrete	Including	Extending	Starting	Stopping	Generalizing	Children
1	CT3	getLocation	Alarm	Call taker	√			1		√						
2	CT2	receiveCall	Alarm	Call taker	√		√		√							
3	PS1	getPosition	Available vehicle list	Positioning system		√	√			√			√			
4	O1	acceptCall	Call taker	operator	√		√		√							
5	I1	createRecord	Call taker	incident	√		√		√							
6	IWL5	getIncident	Call taker	Incident waiting list		√	√		√							
7	I2	getIncidentLocation	Call taker	incident		√		√		√						
8	I3	getIncidentType	Call taker	incident		√		√		√	√					
9	A1	getType	Call taker	alarm		√		√	√		√					
10	I4	prioritize	Call taker	Incident	√		√		√						√	
11	ADB1	search	Call taker	Alarm database	√			√	√							
12	I5	submitRecord	Call taker	incident	√		√		√						√	
13	I6	Update	Call taker	incident	√		√		√							√
14	R9	allocateClass	class of Response	Response	√		√		√						√	
15	UR4	accept	Dispatcher	Unit request		√	√		√							
16	R7	addVehicle	Dispatcher	Response	√		√		√							
17	R4	confirmNumOfVehicles	Dispatcher	Response		√	√		√							√
18	R5	confirmTypeOfVehicles	Dispatcher	Response		√	√		√							√
19	M1	Create	Dispatcher	Mission	√		√		√							
20	RC1	CreateResponse	Dispatcher	Response creator	√		√		√							
21	RP1	findClosestVehicle	Dispatcher	Route planner	√		√			√					√	
22	RP4	findClosestVehicle	Dispatcher	Route planner	√			√		√					√	
23	FR11	receive	Dispatcher	Final report	√		√		√							
24	UR5	reject	Dispatcher	Unit request		√	√		√							
25	CR1	Select	Dispatcher	Class of Response	√		√		√						√	
26	I17	select	Dispatcher	incident	√		√		√							
27	M2	update	Dispatcher	mission	√		√		√							√
28	RMS1	receiveReport	Final report	Record management sys.		√		√	√							
29	R8	updateStatus	Final report	Response		√		√		√						
30	IWL1	addIncident	Incident	Incident waiting list		√	√		√							
31	IRL2	deleteIncident	incident	Incident Response list		√		√	√							
32	IWL3	deleteIncident	incident	Incident waiting list		√	√		√							
33	CT4	getIncidentInfo	incident	Call taker		√		√	√		√					
34	RMS3	receiveIncident	incident	Record management sys.		√		√	√							
35	IWL4	Sort	Incident waiting list	Incident waiting list	√		√		√						√	
36	PP1	getAssistInfo	mission	Primary Police		√		√	√							
37	PV1	receiveInfo	Mission	Police vehicle		√		√		√	√		√			
38	I7	createRecord	operator	Incident	√		√		√							
39	I15	getIncidentDescription	operator	Incident		√	√			√						
40	I14	getIncidentExtent	operator	Incident		√		√	√			√				
41	I13	getIncidentFire	operator	Incident		√	√		√							
42	I12	getIncidentInjury	operator	Incident		√	√		√							
43	I9	getIncidentLocation	Operator	Incident		√		√		√						
44	I11	getIncidentStatus	operator	Incident		√		√		√	√					
45	I10	getIncidentTime	Operator	Incident		√		√	√							
46	I8	getIncidentType	Operator	Incident		√		√	√		√					
47	I16	submitRecord	operator	incident	√		√		√							√
48	AVL3	addVehicle	Police officer	Available vehicle list		√		√	√							
49	ODL1	addVehicle	Police officer	On duty list		√	√		√							
50	AVL1	deleteVehicle	Police officer	Available vehicle list		√		√	√							
51	ODL2	deleteVehicle	Police officer	On duty list		√	√		√							
52	PO1	getMission	Police vehicle	Police officer		√	√		√							

Volume: 08 Issue: 05 Pages: 3188-3200 (2017) ISSN: 0975-0290

	Label	Meaning	Source Object	Destination Object	High Level	Low Level	Primary	Secondary	Essential	Concrete	Including	Extending	Starting	Stopping	Generalizing	Children
53	FR1	Create	Primary Police	Final report	√		√		√							
54	IR2	create	Primary Police	incident report	√		√		√							
55	UR1	create	Primary Police	Unit request	√		√		√							
56	UR2	determineNumOfVehicles	Primary Police	Unit request		√		√	√							
57	UR3	determineTypeOfVehicles	Primary Police	Unit request		√	√			√						
58	FR10	finish	Primary Police	Final report	√		√		√					√		
59	M3	Finish	Primary Police	mission	√		√		√					√		
60	FR4	getArrivalTime	Primary Police	Final report		√	√		√							
61	FR9	getAssistUnits	Primary Police	Final report	√			√	√							
62	FR2	getDate	Primary Police	Final report		√		√	√							
63	FR5	getDispatcher	Primary Police	Final report	√		√		√							
64	FR7	getHowReceived	Primary Police	Final report		√	√			√						
65	FR6	getMission	Primary Police	Final report		√	√			√						
66	FR8	getPrimary Police	Primary Police	Final report	√		√		√							
67	PO2	getStatus	Primary Police	Police officer		√		√	√		√		√			
68	FR3	getTime	Primary Police	Final report		√		√	√							
69	IR1	update	Primary Police	Incident report	√		√		√							
70	CT1	receiveCall	reporter	Call taker	√		√		√					√		
71	I19	Close	Response	incident	√		√		√							
72	RMS2	receiveResponse	Response	Record management sys.		√		√		√						
73	IRL1	addIncident	Response creator	Incident Response list		√	√		√							
74	ES2	alert	Response creator	Emergency service	√		√		√							
75	R1	create	Response creator	Response		√	√		√							
76	R2	determineNumOfVehicles	Response creator	Response		√	√		√		√					
77	R3	determineTypeOfVehicles	Response creator	Response		√	√			√						
78	ES1	getIncidentInfo	Response creator	Emergency service		√		√		√						
79	R6	Submit	Response creator	Response	√		√			√				√		
80	RP2	calculateDistance	Route planner	Route planner		√	√		√						√	
81	RP3	compareDistance	Route planner	Route planner		√	√		√						√	
82	I18	getIncidentLocation	Route planner	incident		√	√			√						
83	AVL2	search	Route planner	Available vehicle list	√			√	√			√				
84	MAV1	select	Route planner	Most appropriate vehicle		√	√		√						√	
85	D1	receiveRequest	Unit request	Dispatcher		√	√		√							
		Sum			35	50	57	28	66	19	8	2	3	4	10	5

Volume: 08 Issue: 05 Pages: 3188-3200 (2017) ISSN: 0975-0290

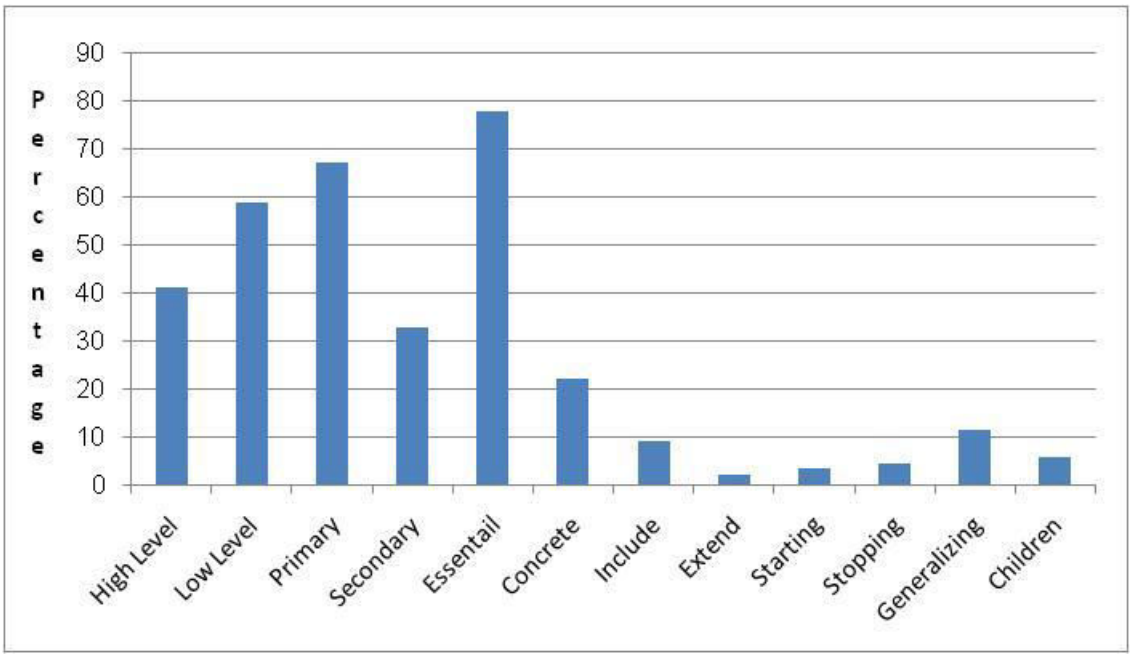

Fig. 5: Number of Use cases identified in the development of Control Command Police System

As we can see in the table, these are 85 events in the control command police system. Each event must be responded by a specific functionally of the objects. The number of Use cases identified in the development of Control Command Police System are depicted in Fig. 5. From the figure, we derive several observations as follows:

- **Observation-1:** The percentages of 'Primary' and 'Essential' use cases in the Control Command Police system almost have the most highest value, respectively. It is because of 'Primary use cases' constitute the reason for which the system developed and 'Essential use cases' make business solutions that are independent of implementation.
- **Observation-2:** The percentage of 'Low Level' use case is significantly more than 'High Level' use cases. It is due to the 'High Level use case' are general activities while 'Low Level use cases' are where we need the specific activities**.**
- **Observation-3:** The percentage of 'Essential' use cases is significantly more than 'Concrete' use cases.It is because of we have not too much use cases concerned with specific hardware such as GPS in the control command police system.
- **Observation-4:** The percentage of 'Primary' use cases is significantly more than 'Secondary' use cases.It is due to the 'Secondary use cases' involves rare and exceptional conditions.
- **Observation-5:** The percentage of 'Including' use case is significantly more than 'Extending' use cases.
- **Observation-6:** The percentages of 'Starting use cases' in the system are more than that of 'Stopping use case' use case, i.e. only one. This relatively higher percentage shows that 'Starting use cases' have more important in the software.

4. GUIDELINES

From a modeling perspective, a use case must capture the series of interactions between an actor and the system that achieves some useful business goal for the initiator of the interaction. The identification of responsibilities of the actors is a good base from which to find reasons for the actor to interact with the system. In this section, several guidelines are recommended to perform any use case modeling, based on our experience obtained from developing use cases in the control command police system.

4.1 Guideline for Identifying High-level, Primary and Essential Use Cases

When dealing with high level, primary and essential use cases, we must identify general activities first. These general activities constitute high-level use cases that are actually defined by a set of low-level use cases. The low-level use cases are where the specific activities are identified. The guidelines for performing this step follow:

- We must not introduce too much detail in the basic descriptions of a use case. It is normal for a description to seem trivial by the time that analyst completes documenting the use case. The value of keeping it simple is to give us a mental nudge when we are bogged down in details later.
- The preconditions section of the use case description should identify what information is required for these use cases to execute normally.
- We must avoid 'technology-dependent use cases' like load, save, startup, and shutdown when identifying 'High Level', 'Primary' and 'Essential' use cases because addressing business use cases still does not finish. There is not

Volume: 08 Issue: 05 Pages: 3188-3200 (2017) ISSN: 0975-0290

enough information available to effectively identify appropriate behavior. These use cases will be the last ones to be developed because we must wait until sufficient details are known to identify what information must be initialized during startup and preserved during shutdown.

4.2 Guidelines for developing Secondary, Concrete and Low-level Use Cases

When dealing with secondary, concrete and low-level use cases, we are introducing details that border on design. The development of these use cases should only be attempted by individuals with significant design skills. An extremely common problem encountered is for a poor design to be specified in these use cases. We must look into the structure of the system rather than the system itself. The guidelines to following during the early iterations are as follows:

- If the 'detail section' of a use case includes another use case, we must identify all of the exceptions that the included use case can throw for that step. This allows the 'including use case' to identify the error condition to which it must react. Of course, these exceptions will be identified in the exceptions section of the description for the 'included use case'.
- For each step in the 'details section', we must identify what errors or alternatives can occur. Each error is examined in terms of what actions should be taken to keep the model consistent. The information necessary to identify what actions should be taken is often clear from the context in which the error occurs.
- We must not put a lots of 'ifs' and 'go to/jumps' in the 'details section' because they interfere with understanding the domain.
- It is important to label each step appearing within the 'details section' of the template (Table 1) with a number. This allows us to make cross-references to that step in other sections of the use case. This is extremely significant when it comes to identifying 'exceptions' and 'constraints' of the template.
- We must capture 'exceptions' in a table that includes three columns: (a) the step in which the error occurs, (b) a label for the error, and (c) the actions that should be performed. As was the case with the details section, it is useful to number each step in the actions to be performed.

4.3 Guidelines for developing Generalizing, Children, Including and Extending Use Cases

When dealing with 'including', 'extending', 'generalizing' and 'children' use cases, the following guidelines must be considered:

- We must view 'including use case' as a relation that identifies a use case that acts like a subroutine to other use cases. Typically, included use cases will not have actors that initiate them. We can consider these use cases as inheriting actors.
- In some cases, a number of use cases all share a common structure with the exception of some minor additional steps. These cases can be simplified as an extension of a common 'generalizing use case'. In this case, the use case exploits the details of another use case and identifies where the additional details are incorporated.
- The precondition section of a use case that extends another identifies the condition that determines if the 'extending use case' should be invoked.
- In some cases, the same general activity may take place in several use cases, but have significantly different details depending upon the entities that participate in them. Even though 'generalizing use case' should have been identified earlier than this point, it is still a good idea to examine the use cases to determine if new 'generalizing use case' can be added.

4.4 Guidelines for Developing Starting, Stopping, Frond End and Back End Use Cases

When dealing with 'Starting, 'Stopping', 'Frond End' and 'Back End' use cases, the following guidelines must be considered: The most common situation encountered among people writing use cases for the first time is that they immediately start writing use cases for starting and stopping the system. The main problem is that they don't even know what the system is to do, yet they are worried about what initialization activities have to take place. The guidelines to establish when these use cases should be developed are as follows:

- In writing 'Frond End' Use case, we must be worrying about screens that lead to difficulty in writing the use cases. Often, analyst gets about halfway through the use case and then starts describing what some screen looks like. The description can go on for pages if the screen layout is complex. Instead, the analyst should only identify the objects present on the screen. Even then, only focus on those that apply to the use case. We must not worry about the layout of the buttons and fields on the screen. It is the interaction with the screen that is important in the use case. That can be done as a figure in the 'comment section' of the 'use case description' or in a separate description from the use case.
- In writing 'Starting' Use case, we must consider that the initialization activities are highly design-dependent. If the analyst develops 'essential' use cases, then there will not be sufficient information to identify what actions should be performed during 'starting' and 'stopping' use cases'. These events should not be developed for 'essential use cases'.

Volume: 08 Issue: 05 Pages: 3188-3200 (2017) ISSN: 0975-0290

- When the analyst develops 'concrete use cases', the 'Starting' and 'Stopping' use cases should only be addressed once all of the 'essential' and 'secondary' use cases have been developed. At this point, one has sufficient detail to identify if connections to external actors should be created during initialization or not; whether specific structural details have to be constructed, and so on.
- It is important to distinguish the service request or event notification the actor is initiating from the manner (action) in which the actor invokes the request or event notification. In many cases, the same service request can be invoked in multiple fashions: by keystrokes, menu items, or buttons. However, the resulting activities of the system are identical. It is this later component that we are attempting to capture in use cases.

5. SUMMARY AND CONCLUSION

This paper reviewed different use cases as a means for use case modeling in software development. From a modeling perspective, a use case must capture the series of interactions between an actor and the system that achieves some useful business goal for the initiator of the interaction. The identification of responsibilities of the actors is a good base from which to find reasons for the actor to interact with the system. The recommended approach is:

a) We must develop a primary, essential, high-level use case model and consider scenarios as instances of use cases. Creating more abstract scenarios to develop use case is necessary if two or three scenarios look very similar. We must be cautious of creating more than 40 use cases to cover the fundamental system actions, 'High Level' use cases (It was 35 in our experiment, as a large system). Additional use cases for unusual events should be chosen with care and kept to a manageable number.

b) If the business domain is not well understood, we must use the model of the step (a) to develop a primary and secondary, essential, low-Level use case model. To identify details, we must start simple and slowly introduce complexity. We must focus first on the simple case where everything is perfect and no problems exist. It is not a bad idea to give a very brief set of details initially for each use case, focusing only on course features. This allows analyst to identify supporting use cases that simplify the process by extracting common details into other use cases.

c) If the technology is not well clarified, we must use the model of the step (b) to develop a primary, concrete, low-level use case model.

d) If the system involves with reliability, we must use the models of the step (b) and the step (c) to help develop a secondary, concrete, low-level use case model.

REFERENCES

[1]. N. Ashrafi, H. Ashrafi, *Object Oriented Systems Analysis and Design*, Pearson, 2009.

[2]. S.H. Pfleeger, and J.M. Atlee, *Software Engineering: Theory and Practice*, 4th Edition, Pearson, 2010.

[3]. R.S. Pressman, *Software Engineering: A Practitioner's Approach*, 8th Edition, McGraw-Hill, 2015.

[4]. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall, 1992.

[5]. D. Rosenberg, and M. Stephens, *Use Case Driven Object Modeling with UML: Theory and Practice*, Apress, 2007.

[6]. C. Larman, *Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd edition, Prentice Hall, 2005.

[7]. J. Rumbaugh, *Getting Started: Using Use Cases To Capture Requirements*,Object-Oriented Programming, September, 1994.

[8]. I. Jacobson, G. Booth, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[9]. A. Cockburn, *Writing Effective Use Cases* (Draft 3), Addison Wesley Longman, 2000.

[10]. B. Bruegge, and A.H. Dutoit, *Object-Oriented Software Engineering: Using UML, Patterns, and Java*, Pearson Prentice Hall, 2010.

[11]. R.C Lee, and W.M. Tepfenhart, *UML and C++: A Practical Guide to Object-Oriented Development*, 2nd Edition, Pearson Prentice Hall, 2005.

[12]. P. Coad, and E. Yourdon, *Object-Oriented Analysis*, Yourdon Press, 1991.

[13]. N. Goldsein, and J. Alger., *Developing Object-Oriented Software for the Macintosh Analysis, Design, and Programming*, Addison-Wesley, 1992.

[14]. M. Langer, *Analysis and Design of Information Systems*, 3rd Edition, Springer-Verlag London Limited, 2008.

[15]. J. Martin, and J. Odell, *Object-Oriented Analysis and Design*, Prentice-Hall, 1992.

Volume: 08 Issue: 05 Pages: 3188-3200 (2017) ISSN: 0975-0290

[16]. Y. Sommerville, *Software Engineering*, 9th Edition, Pearson Education, 2010.

[17]. H. Rashidi, *Software Engineering - A programming approach*, 2nd Edition, Allameh Tabataba'i University Press (in Persian), Iran, 2014.

[18]. G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[19]. J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

[20]. M. Fowler, and K. Scott, *UML Distilled A Brief Guide to The Standard Object Modeling Guide*, 2nd Edition,Addison Wesley Longman, Inc, 1999.

[21]. I. Jacobson, M.P. Christerson, and F. Overgaard, *Object-Oriented Software Engineering - A Use Case Approach*, Addison-Wesley, Wokingham, England, 1992.

[22]. G. Bavota, A.De. Lucia, A. Marcus, and R. Oliveto, *Automating extract class refactoring: an improved method and its evaluation, Empirical Software Engineering*, Vol. 19, pp. 1616-1664, 2014.

[23]. G. Canforaa, A. Cimitilea, A.D. Luciaa, and G.A.D. Lucca, Decomposing Legacy Systems into Objects: An Eclectic Approach, *Information and Software Technology*, Vol. 43, pp. 401-412, 2001.

[24]. A.V. Deursen, and T. Kuipers, Identifying Objects Using Cluster and Concept Analysis, *Proc. of 21st International Conference on Software Engineering*, Los Angeles, CA, ACM Press, New York, pp. 246-255, 1999.

[25]. M. Fokaefs, N. Tsantalis, E. Strouliaa, and A. Chatzigeorgioub, Identification And Application Of Extract Class Refactoring In Object-Oriented Systems, *Journal of Systems and Software*, Vol. 85 , pp. 2241–2260, 2012.

[26]. J.V Gurp, and J. Bosch, *Design, Implementation, and Evolution of Object-Oriented Frameworks: Concepts and Guidelines, Software, Practice and Experience*, Vol. 31, pp. 277-300, 2001.

[27]. H. Rashidi, Objects Identification in Object-Oriented Software Development - A Taxonomy and Survey on Techniques, *Journal of Electrical and Computer Engineering Innovations*, Vol. 3 (2), pp. 27-43, 2015.

[28]. S. Schlaer, and S. Melior, *Object Lifecycles: Modeling the World in States*, Yourdon Press, 1992.

[29]. R. Wirfs-Brock, *Designing Object-Oriented Software*, Prentice-Hall, 1990.

[30]. M. Josuttis Nicolai, *The C++ Standard Library: A Tutorial and Reference*, Addison-Wesley, 1999.