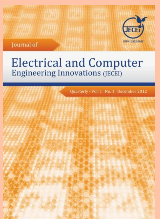
SRTTU

Journal of Electrical and Computer Engineering Innovations

JECEI, Vol. 3, No. 2, 2015

Regular Paper



Objects Identification in Object-Oriented Software Development - A Taxonomy and Survey on Techniques

Hassan Rashidi

Department of Statistics, Mathematics, and Computer Science, Allameh Tabataba'i University

Corresponding Author's Information: hrashi@atu.ac.ir

ARTICLE INFO

ARTICLE HISTORY:
Received 8 July 2015
Revised 23 December 2015
Accepted 25 December 2015

KEYWORDS:
Taxonomy
Class
Object
Object-Oriented
Software Engineering

ABSTRACT

Analysis and design of object oriented is one modern paradigms for developing a system. In this paradigm, there are several objects and each object plays some specific roles. Identifying objects (and classes) is one of the most important steps in the object-oriented paradigm. This paper makes a literature review over techniques to identify objects and then presents six taxonomies for them. The first taxonomy is based on the documents exist for a domain. The second taxonomy is based on reusable previous knowledge and the third one relies on commonalities in a domain. The fourth taxonomy is concerned with decomposing a domain. The fifth taxonomy is based on experience view and sixth one is related to use the abstraction in a domain. In this paper, the constraints, strengths and weaknesses of the techniques in each taxonomy are described. Then, the techniques are evaluated in four systems inside an educational center in a university. A couple of approach is recommended for finding objects, based on some practical experiences obtained from the evaluation.

1. Introduction

Object-Oriented (OO) is one modern paradigm for developing software. In this paradigm, we describe our world using the object categories (classes) or object types (pure abstract class or Java interface) (see[18],[19],[23],[24], [31], [42], [43], [46]). Each class/object plays a specific role in the software. These roles are programmed in Object-Oriented languages such as C++ and Java.

Several attributes (data variables) and services (operations/functions/methods) are assigned to these classes. Then, we model the behavior of the world as a sequence of messages that are sent between various objects. In OO models, a number of relationships (inheritance, association, and aggregation- see [11], [14], [38],[45] and [46]) are identified between the classes/objects. Moreover, there are many popular design modeling processes and guidelines such as GRASP [49] and ICONIX [48] for assigning responsibility to classes and objects in object-oriented design.

The first step for building an OO model is to find out the objects, on which we focus. In this step, we are not really finding objects. In fact, we are actually finding categories and types (analysis concepts) that will be implemented using classes and pure abstract classes. The results of problem analysis is a model that: (a) organizes the data into objects and classes, and gives a structure to the data via relationships of inheritance, aggregation, and association; (b) specifies local functional behaviors and defines their external interfaces; (c) captures control or global behavior; and (d) captures constraints (limits and rules).

The main motivation of this paper is to have a survey on the techniques to find the potential objects and makes six taxonomies for them. The remainder of this paper is as follows. In Section 2, the literature review and taxonomies of techniques to find objects are presented. In Section 3, the experiences of applying the approaches to four systems are presented. In Section 4, two approaches to find objects in the object-oriented paradigm are recommended. Finally, Section 5 is considered for summary and conclusion.

2. Literature Review and Taxonomies

One of the major challenges in the Object-Oriented development and transforming the legacy systems into Object-Oriented one is how to identify objects. To do this, many methodologists have their own favorite techniques. Almost, all techniques have shortcomings; i.e., they sometimes fail to identify all objects and sometimes identify false objects. Deursen and Kuipers (1999) have used clustering and concept analysis to identify objects in the legacy code [15]. Canfora et al. (2001) have employed an eclectic approach to decompose legacy systems into objects [10].

Few researches have focused on refactoring the systems and extract class in this area. Fokaefs et al. (2012) have described a method and a tool, designed to fulfill exactly the extract class refactoring [17]. It has three steps: (a) recognition of extract class opportunities, (b) ranking the opportunities in terms of improvement to anticipate which ones should be considered in the system design, and (c) fully automated application of the refactoring selected by the developer. The first step relies on a hierarchical agglomerative clustering algorithm based on Jaccard distance between class members, which identifies cohesive sets of class members within the system classes. The second step, measures the design quality by the entity placement metric. Through a set of experiments, implemented as an Eclipse plug-in, the research has shown that the tool is able to identify and extract new classes that developers recognize as "coherent concepts" and can improve the design quality of the underlying system.

Bavota et al. (2014) have proposed an approach for automating the extract class refactoring [1]. This approach analyzes structural and semantic relationships between the methods in each class to identify chains of strongly related methods. The identified method chains are used to define new classes with higher cohesion than the original class, it can also preserve the overall coupling between the new classes and the classes interacting with the original one. In the literature, there are several reported works in which the objects, in the object-oriented software, are classified. Jacobson et al. in [21] and [22] categorized the objects into Entity, Boundary, and Control. The Entity objects represent the persistent information tracked by the system. The Boundary Objects represent the interactions between the actors and the system. The Control objects represent the tasks that are performed by the user and supported by the system. Coad and Yourdon ([12], [47]) categorized the objects into different groups: (a) Structure ("kind-Of" and "part-Of" relationships); (b) Other systems (External Systems); (c) Devices; (d) Events (A historical event that must be recorded); (e) Roles (the different roles that are applied to the users); (f) Locations; (g) Organizational units (groups to which the user belongs).

Schlaer and Mellor in[39] and [40] categorized objects into five groups: (a) Tangibles (cars, telemetry, sensors); (b) Roles (mother, teacher, and programmer); (c) Incidents (landing, interrupt, collision); (d) Interactions (Loan, meeting, marriage); (e) Specification (product specification, standards).

Ross [36] categories objects into six groups: (a) People (humans who carry out some function); (b) Places (areas set aside for people or things); (c) Things (physical object); (d) Organizations (collection of people, resources, facilities, and capability having a defined mission); (e) Concepts (principles or ideas not tangible, per se); (f) Events (things that happen-usually at a given date and time, or as steps in an ordered sequence. One major gap and research need is to have an overview and taxonomy on techniques to identify objects in Object-Oriented software development. According to Merriam-Webster[29], taxonomy is the study of the general principles of scientific classification, and is especially the ordered classification of items according to their presumed natural relationships. The major difference between techniques to find out objects, in general, depends on the circumstances around existing some documents in a domain, how previous knowledge are reused, how the commonalities are factored out, how the composition of a domain is performed, how the experience of developers aids, what level of abstraction is used, and how we use individual objects. There are, therefore, six taxonomies to categorize techniques that find out objects in Object-Oriented development. They are described in the following with their advantages and disadvantages.

A. The first taxonomy: Document view

for identifying objects is concerned with existing document such as the requirement analysis report or the data flow diagram in a domain. Therefore, there are a couple of paradigms for this taxonomy such as, using nouns and using data flow-diagram:

Use Nouns (UN):

This technique is traditional and starts with the document written for a problem. It was invented by Russell J. Abbott and popularized by Grady Booch ([4], [5], [6]) and cited in many publications (e.g. [26], [32], [33] , [41], [44]). To use this technique, the nouns, pronouns and noun predicated in the written documents are used to identify objects. This technique has many advantages: (i) Narrative language (English, Chinese, French, German, Japanese, etc.) are well understood by everyone in a project staff; (ii) there is usually one-to-one mapping from nouns to objects or classes; (iii) Using nouns requires no learning curve; the technique is straightforward and well defined, and does not require a complete paradigm shift to the OO paradigm for the beginner; (iv) This technique does not require a prior *Object-Oriented Domain Analysis* (*OODA*); the analyst can apply it to an existing requirement specification [37], written for structural analysis and / or any other methodology. On the contrary of the advantages, this technique has some shortcomings, in general. For one thing, this is an indirect approach to find objects and classes. Nouns are not always classes or objects in problem domain. In many cases, the nouns, especially subjects of sentences, refer to: (a) an entire assembly or a computer software configuration; (b) a subassembly or a software component; (c) an attribute; (d) a service.

Use Data Flow Diagrams (UDFD):

This technique was first published by Seidewitz and Stark of NASA's Goddard Space Flight Center [26]. It assumes that a *Data Flow Diagram (DFD)* in the domain exists. The major benefit of this technique is that it requires no paradigm shift by the analysts and developers. If the original DFDs are well constructed, false-positive identification of objects and classes are rare. Additionally, there are a lot of projects that already have the context diagrams and DFDs. Unfortunately, the shortcoming of UDFD is also directly related to not making the paradigm shift. Nearly all of the DFDs were originally written for functional decomposition, and they have a tendency to create a top-heavy architecture of classes. With functional decomposition, there is a tendency to assume that the stem is an assembly of subassemblies at the appropriate level. Developers tend to assign services at the corresponding level where the subassembly was found. This may cause objects to be identified in the wrong subassembly. Although false-positive identification of objects and classes is rare, not all of the objects or classes are identified. The rareness of false-positive identification is totally dependent on the quality of the original DFDs. This is still an indirect method of finding objects and classes; it is based on data abstraction and not on object abstraction. In many instances, an object or class contains more than one data store. Thus, their attributes may be mapped to objects and classes while their associated objects and classes remain unidentified. Because the DFDs represent functional decomposition, pieces of an object may be scattered across several DFDs assigned to different persons. Thus, different variants of the same object may be redundantly and independently identified. Transforms are not required to be a service of an object. Therefore, transforms are often compound operations that need to be assigned to multiple objects. If the objects are not properly identified, this leads to fragmented objects and classes.

B. The Second Taxonomy: Knowledge View

The second taxonomy is based on reusing previous knowledge from which objects are explicitly extracted. The previous knowledge can be collected already in the Object-Oriented domain analysis, framework, repository and individual objects(classes). There are four techniques in this taxonomy:

Use OO Domain Analysis (UOODA):

This technique is specified in [26] ,[33] ,[34] and [35]. This technique assumes that an OO Domain Analysis has already been performed in the same problem domain. This technique supports the reuse and tends to maximize the cohesion in classes and minimize the message and inheritance coupling. If one assumes that the previous OODA is solid, indeed this technique offers a "reality check" on present work because the objects and classes should be the similar to the ones in the OODA. Thus, considerable time and effort can be saved if the original OODA is relevant and complete. On the contrary, finding adequate and relevant OODA is not easy today. Most systems have either incomplete OODA or no OODA model at all. To make the reuse more effective, the problem domain must be well documented and understood by the developers. Tailoring for performance and other business constraints in a specific project may decrease the reuse. Although it is easier to reuse than to reinvent, the Not-Invented-Here (NIH) syndrome of many developers must be successfully overcome.

Reuse an Application Framework (RAF):

This technique is specified in [20], [26] and [35]. Gurp et al. (2001) defined it as a partial design and implementation for an application in a given domain[20]. This approach assumes that at least one OODA has been already performed to create an application framework of reusable classes. RAF has some limitations. Developers must be able to identify one or more relevant application frameworks that have been previously developed and stored in a repository. Most likely, not all of the needed classes will be in the application framework(s) examined. One concern with application frameworks is the NIH syndrome. This syndrome is translated into a general belief that if the application framework was not developed locally, then it cannot take into account all of the concerns of the local team. This concern is not totally unfounded. In particular, application frameworks often contain both analysis and design classes. Unfortunately, it is not easy to distinguish between these two types.

Reuse Class Hierarchies (RCH):

This technique is specified in[26] ,[33] and [35]. This technique assumes that a reuse repository with relevant reusable class hierarchies has been developed. This technique has the same advantages as using OODA. The major advantage of this technique is that it maximizes the use of inheritance and is a natural fit for some OO languages. In contrast, it has additional limitations beyond those for OODA as with all techniques. Additionally, the existing classification hierarchies may not be relevant to the current application. Existing classes may need to be parameterized, or new subclasses may need to be derived.

Reuse Individual Objects and Classes (RUIOC):

This technique is specified in [26] ,[33] and [35]. We can reuse specific objects and classes from the repository with relevant reusable objects and classes. The major advantage of this technique is that it is inexpensive and easy to use so that little efforts is invested in making the classes, and they can be easily discarded. Also, this method stimulates communication and is not intimidating to beginners. On the contrary, this technique has some very serious shortcomings. One major concern with the repository is NIH syndrome. This syndrome is translated into a general belief that if the repository is not developed locally, then it cannot take into account all of the concerns of the local team.

C. The Third Taxonomy: Commonalities View

The third taxonomy for identifying objects is based on finding communalities and factoring out on them. In this view, we have a couple of techniques, which are specified in the following:

Use Generalization (UG):

This technique is specified in[26] and [33]. It assumes that objects are identified prior to their classes (every object is an instance of some class), and that communalities among objects can be used to generalize classes. The first advantage of this approach is that it promotes reuse and supports the development of one or more classification hierarchies. In contrast, UG requires significant training, practice, intuition, and experience.

Use SubClasses (USC):

This technique is specified in [26] and [35]. The steps are: (a) Identifying classes that share common resources (i.e., attributes, service name, methods, etc.); (b) Factoring out the common resources to form a super-class (parent), and then use inheritance for all classes that share these resources to form simpler subclasses. When using subclasses, we skip finding objects and directly start identifying classes. The key benefit of this technique is reuse. In contrast, when misused, it leads to difficult maintainability and opaque classes that reuse randomly unrelated resources that do not logically belong to subclasses of the same superclass. Additionally, USC also may produce inappropriate or excessive inheritance coupling.

D. The Fourth Taxonomy: Decomposition View

The fourth taxonomy for identifying objects is based on decomposition view; i.e., how we decompose a domain and its objects. In this view, we have a couple of techniques, which are specified in the following:

Use Subassemblies Method (USM):

This technique is specified in [26] ,[33] and [35]. It assumes that the developers are incrementally developing subassemblies using a recursive development process. The major advantage of this technique is that it supports incremental identification of objects/ classes. It also identifies all the subassemblies in an application domain. It is very similar to functional decomposition ([32], [33]), so there is less culture shock for developers trained in the structured methodology. On the contrary, there are some limitations for implementation of this technique. It identifies only assembled objects. Thus, one must have some other techniques to identify fundamental components of the subassemblies.

Use Object Decomposition (UOD):

This technique is specified in [26] and [35]. This technique assumes that most objects are composed of the other objects. The key benefit of this technique is reuse, but it has some serious drawbacks. When misused, it leads to un-maintainable and opaque classes that reuse randomly unrelated resources that do not logically belong to subclasses of the same superclass. It also may produce inappropriate or excessive inheritance coupling.

E. The Fifth Taxonomy: Experience View

The fifth taxonomy for identifying objects is based on how we use personal experience in different human activity. In this view, we have a couple of techniques, which are specified in the following:

Use Personal Experience (UPE):

UPE technique is presented in [26] and [35]. This technique assumes that the developer has already performed an analysis and can use its experience. Based on one's experience, this technique provides a reasonable "reality check" on projects. Thus, the quality of the classes and objects may be substantially improved, as they are based on classes and objects that are already built and tested. It is also very common to want to leverage off the application experience of the developer. This technique considers the relevant previous experience, which is not always available. AS a disadvantage of UPE, that the past experience may be of limited value and may possibly even be misleading. Moreover, this technique is very informal, and different developers may identify substantially different objects and classes given the same starting information

Use Class-Responsibility-Collaboration cards (UCRC):

This technique was developed by Beck and Cunningham. It is based on this fact that identifying the objects and classes is a human activity that can be stimulated by use of small pieces of paper to represent objects/classes [33]. This technique is inexpensive and easy to use. Little attempt is invested in making the classes, and they can be easily discarded. Also, this method stimulates communication and is not intimidating to beginners. Historically, this method is more suitable for thinking about and designing the objects and classes rather than identifying them. The major disadvantage of UCRC is that software engineers must already have objects and classes in order to use this technique to identify additional objects and classes. The developers must have significant experience, creativity and intuition for this technique to be consistently successful. However, the revised version based on the use cases described previously is very effective [13].

F. The Sixth Taxonomy: Abstraction View

The sixth taxonomy for identifying objects is based on how we use abstraction in a domain. In this view, we have a couple of techniques, which are specified in the following:

Definitions of Objects and Classes (UDOC):

This technique is specified in [26] and [35]. The technique is very simple; the developer uses object abstraction and the definition of classes to intuitively identify them. This is the same way that the experienced developers would recognize functional and process abstractions. It provides the best partitioning of the requirements into classes. When this method is used properly, it can produce the fewest false-positive identifications. UDOC has no limitations, but it requires a significant paradigm shift for the developer. This paradigm shift requires significant training, practice, intuition, and experience, which usually takes at least 6 months of on-the-job training[26]. There are no tricks or tools to help in this technique; the tools are designed only to document the results.

Use the Things to Be Modeled (UTBM):

This technique is specified in [25], [26] and [35]which explicitly determines the objects/classes. The basic steps in this technique are: (a) Identifying individual or group things, such as persons, roles, organizations, logs, reports, forms, etc. in the application domain; and (b) Identifying the corresponding objects and classes. The major advantage of UTBM is that this technique is highly effective because it is natural, direct, and reliable. On the contrary, it requires significant experience with Object-Oriented to apply successfully. Unfortunately, UTBM tends to help only in finding the terminators and other tangible objects that are the easiest entities to identify. Abstract classes are not readily identified using this technique. Furthermore, this technique requires that the user makes the paradigm shift to the object-oriented mindset. Although this paradigm shift should be the ultimate goal, on-the-job training may be very expensive.

Table-1 makes a summary of the techniques in the taxonomies discussed above. The major assumptions, strengths and weaknesses of the techniques are presented in the columns 3, 4, 5 of the table, respectively.

3. Practical Experience and Guidelines

In order to evaluate the techniques to find objects in practice, two groups of Bachelor and Master students in software engineering were dedicated to perform the techniques under supervision of a couple of experts in Qazvin Islamic Azad University, in the educational center [16]. The students were assigned to do the techniques based on their experiences. In the group one in which there were 20 students, there was little experience in object-oriented development and the students had to do the jobs as the final projects. The group two, 16 Master students, had more experiences and they had to do the jobs as a project in the advanced software engineering course. The students in each group were divided into teams with having 2 students in each. Both groups used the MFC (Microsoft Foundation Classes) as application framework for MS Windows and the reuse repository. Moreover, the groups utilize a *Control Command Police System* (CCPS) role, here, is to act as the given OODA with several reusable classes (see the assumptions of the techniques). A mini-requirement for CCPS is briefly described in [41] and then the system is expanded in [35]. This police service system must respond as quickly as possible to reported incidents. The main objective of this system is to ensure that incidents are logged and routed to the most appropriate police vehicle.

The full specification of the system and its implementation are given in [35]. Due to its fertility for reusability in both application and system software, we selected CCPS in our study where its class diagram is depicted in Figure 1. In this class diagram, there are many classes. The main classes, here, are "Incident", "Police Staff", "Police Vehicle", "Police Officer" ,"Director" ,"Route Manager" ,"Incident Waiting List", "Response", "GPS Receiver" and so on. The following applications were considered as the problem domains on which the groups did their jobs:

- **ATM System:** This system was a simple ATM in which we expected to see use cases covering the principal functions such as withdraw cash, display balance, print statement, deposit cash and change PIN. Description of this case had to be described the actors involved, the inputs and outputs, normal operation and exceptions. More details on this application are given in [41] and [46]. The general class diagram for ATM system is depicted in Figure 2. The experts use this diagram in evaluations of the techniques for identifying the objects.
- **High School System**: In this system, the students wanted to design a software that is a part of the common processes in a non-public school. These processes include registration, classification of students, evaluation of students and teachers discipline, grades, transcripts delivery and financial management of receipts and payments. The overall scenario is as following: The director, teacher, principal and school counselor, as the main responsible, have common roles for student registration processes and training to help parents. In the meantime, all student data are recorded in their files. In the registration for each student, the courses are grouped in the curriculum and can be selected. Then, the student fees are calculated on the basis of their choices. The exams are designed using a question bank that is available to teachers and after grading the results are stored in the file and also offered to the parents. Moreover, this system calculates payroll for the teachers and store them. The general class diagram for this high school system is depicted in Figure 3. The experts use this diagram for evaluating the techniques and the comparisons have done.
- **Voicemail System:** This system was a voice mail system consists of a speaker, a keypad, and a microphone. The students model the operation of an embedded software system for a voicemail system included in a landline phone. This had to display the number of recorded messages on an LED display and should allow the user to dial-in and listen to the recorded messages. To define different levels of access, the individual user information is recorded in the system and each person has their own mailbox to receive and send messages from/to others. More details on this application are given in [41]. The general class diagram for this system is depicted in Figure 4. The experts use this diagram in evaluations of the techniques and comparisons done.
- **Firm Planning System**: In this system, a time series data including balance sheet, profit and loss account, financial ratios, production lines information and other variables relating to personnel, etc. of a firm (company) are available and must be stored in a database. An economic expert helps to estimate several equations to make a model among the time series data. The system must be able to accept several exogenous variables that are imposed from outside the system. The system uses the model to predict the endogenous variables in the coming years according to the equations subject to the exogenous variables. More details on this system are given in [50]. The general class diagram for this system is depicted in Figure 5. The experts use this diagram in evaluations of the techniques and comparisons done.

In order to have a good design, the experts applied several rules to design the class diagrams in Figures 3 to 6. In fact, Classes/Objects must be considered as any real-world entity and they are important to the discussion of the requirements. In summary, the following rules are used to determine the identified objects in the systems:

- **Rule-1(Coherency):** A class should be coherent and simple. A class describes a group of objects with identical attributes, common behaviors, common relationships, and common semantics.
- **Rule-2 (Serviceability):** A class/object must provide some services to other objects (clients). In fact, each object does some functions and each class has at least one public method.
- **Rule-3 (Modularity):** A design is modular when each activity of the system is performed by exactly one class and when the inputs and outputs of each class/object are well-defined. A class/object is well-defined if its interface accurately and precisely specifies its externally visible behavior.
- **Rule-4(Abstraction):** An object is an abstraction of something in a problem domain and has a crisply defined boundary, reflecting the capabilities a system to keep its information, or interact with it, or both of them.

Obviously, just using a single technique is not enough to identify all the objects since each technique has its own assumptions and constraints. Moreover, the true objects were not identified once. Hence, we decided that the groups do their jobs for a couple of rounds in order to show the techniques are trainable and roles of the experts are important. In the first round, we explained the steps for performing each technique (described in Sections 2.1 to 2.6). In this round, the students had to their own understandings of the techniques without any consultations to find the potential objects. In the second round, the students did their jobs by interactive dialog with the experts and more potential objects in the problem domain were identified. At the end of each round, the list of objects identified was recorded. The following definitions and calculations are performed during this study:

- T: Set of Techniques = {UN, UDFD, UOODA, RAF, RCH, RUIOC, UG, USC, USM, UOD, UPE, UCRC, UDOC, UTBM}
- S : Set of Systems = {High school system, ATM system, Voice mail system, Firm planning system}
- NT_1: Number of teams in group 1;
- NT_2: Number of teams in group 2;
- $NOI_{s,1}^{G1,t,n}$: Number of objects identified by team n in system s at the first round of running technique t;
- $NOI_{s,2}^{G1,t,n}$: Number of objects identified by team n in system s at the second round of running technique t;
- $NOI_{s,1}^{G2,t,n}$: Number of objects identified by team n in system s at the first round of running technique t;
- $NOI_{s,2}^{G2,t,n}$: Number of objects identified by team n in system s at the second round of running technique t;

- $ANOI_{s,1}^{G1,t}$: Average number of objects identified in system s by group 1 at the first round of running technique t;
- $ANOI_{s,2}^{G1,t}$: Average number of objects identified in system s by group 1 at the second round of running technique t;
- $ANOI_{s,1}^{G2,t}$: Average number of objects identified in system s by group 2 at the first round of running technique t;

$ANOI_{s,2}^{G2,t}$: Average number of objects identified in system s by group 2 at the second round of running technique t;

$$ANOI_{s,r}^{G1,t} = \frac{\sum_{n=1}^{NT1} NOI_{s,r}^{G1,t,n}}{NT1} \quad r=1,2\,;\ \forall t \in T; \forall s \in S \qquad (1)$$

$$ANOI_{s,r}^{G2,t} = \frac{\sum_{n=1}^{NT2} NOI_{s,r}^{G2,t,n}}{NT2} \quad r=1,2\,;\ \forall t \in T; \forall s \in S \qquad (2)$$

The results of the calculations by (Eq.1) and (Eq.1) for each group in the first and second rounds are shown in Table-2. Note that the numbers in the table are rounded up and the G_iR_r shows the results for group g in round r; g=1,2 and r=1,2. The number shown in the parentheses in front of the name of each system is according to the number of classes in the general class diagram.

At the first glance, we can get the following observation:

- **Observation-1:** Since the number of objects identified by each group of the students in running the second round is more than that of the first one, we can observe that the techniques are trainable and roles of the experts in training are important.

The number of objects identified in each round for the systems is not very convenient to make any judgment because they are absolute value. Hence, we decided to calculate the average percentage of the number of objects identified for both groups of the students. The following equation is used to make the average.

$$AG_s^{Gg,t} = \frac{ANOI_{s,1}^{Gg} + ANOI_{s,2}^{Gg}}{2 * Number\ of\ Classes\ in\ system\ s} \times 100; g = 1,2;\ t = \forall t \in T; \forall s \in S \qquad (3)$$

The results of the calculations by (Eq.3) for each group in four systems are depicted in Figure 6.

Following observations can be obtained From this figure:

- **Observation-2:** The average number of objects identified by the group 2 in running the techniques for each system is more than those of the group 1. It is due to this fact that the students in the group 2 were M.Sc. Students while the students in the group 1 were B.Sc. students.
- **Observation-3:** The average number of objects identified in Firm planning system is less than those identified in the other systems. It is highly due to the students in both groups had not significant experience in the financial and accounting systems.

In order to sum-up the results over each technique, we calculate the average number of objects identified by applying the techniques for both groups of students. The following equations are used to sum-up the averages. Figure 7 and Figure 8 depict the results of the calculations obtained from (Eq.4) and (Eq.5), respectively, As we can see in the figures, the following observations are obtained in our study:

- **Observation-4:** Although the technique UPE is traditional, it is highly effective in our study. This technique is capable to identify about 90 percent of the objects in the systems.

Previons investigations showed that, this technique suggest a foundation for some natural language processing tool in a semantic network [44].

$$AOI_t^g = \frac{ANOI_{s,1}^{g,t} + ANOI_{s,2}^{g,t} g}{\sum_{\forall s \in Systems} 2 * Number\ of\ Classes\ in\ system\ s} \times 100; g = G1, G2;\ t = \ each\ technique\ in\ T \quad (4)$$

$$AO_t = \frac{\sum_{\forall s \in Systems} ANOI_{s,1}^{G1,t} + ANOI_{s,2}^{G1,t} + ANOI_{s,1}^{G2,t} + ANOI_{s,2}^{G2,t}}{\sum_{\forall s \in Systems} 4 * Number\ of\ Classes\ in\ system\ s} \times 100; t = \ each\ technique\ in\ T \quad (5)$$

- **Observation-5:** The techniques UOODA and RAF were able to identify between 70 to 80 percent of the objects.
- **Observation-6:** The techniques UDFD, UCRC, and UTBM have approximately the same results in identifying the number of the objects.
- **Observation-7:** The technique UOD, UG, USC, RCH, RUIOC and USM are not effective in identifying objects in the systems. They could not able to identify more than 50 percent of the objects. This is due to this fact that these techniques need more understanding the concept of the class and objects as well as the hierarchical imagination. It seems that the students minimized the message and inheritance coupling during running these techniques.
- **Observation-8:** Although Table-1 shows that the USM and UOD are effective, they have the lowest affectivity in our experiments. These techniques identified only between 30 to 40 percent of the objects.

Based on the observations obtained from the evaluations, the techniques presented in Section 2 can be classified in two types:

- **Conventional**: Since the techniques UOD, UG, USC, RCH, RUIOC and USM discovered around 50% of the True objects in the systems, we categorizes those as the conventional type.
- **Modern**: Since the techniques UN, UDFD, UCRC, UTBM, UDOC, UOODA, UPE and RAF discovered more than 50% of the objects as the true ones in the systems, we categories those as the modern techniques. Although UN seems as the most traditional techniques, this is widely used and applied to many object-oriented developments. Subhash et al. (2012) used this technique [44] and made a natural language-based tool which aims at supporting the analysis stage of software development in an object-oriented framework.

Regarding the observations and experience obtained in our study, we made some guidelines as follows:

- **Guideline-1**: The personal experience is a highly subjective technique, which is derived from observation 1-4. The messages and inheritance coupling may not be minimized in applications. Uses of various knowledge such as application knowledge, design knowledge and general world knowledge have significant roles in finding objects. In the application knowledge, interviews of developers with end users and experts are performed, to determine the abstractions of the application domain. Design knowledge and general world knowledge employ reusable abstractions in the solution domain and use the generic knowledge and intuition.
- **Guideline-2:** The application frameworks and Reuse Repositories have important roles in finding objects. They help to find out more potential objects in the systems. This guideline is derived from observation-5.
- **Guideline-3:** Formulations of use cases and Scenarios (the instances of use cases) in natural language and UML have significant roles in finding objects. This guideline is derived from observation-6.
- **Guideline-4:** Training and education of the techniques and concepts of terms are very important for applying the techniques correctly to different domains. This Guideline is derived from observation-7 and observation-8. Table-3 shows the possible cases for using terms according to concepts in application domain.
  - **Note-1:** Usually several different nouns, or noun phrases, are used to describe the same thing as concepts or ideas. A single term must be selected, and the alternative one must be eliminated. For example, the words "location" and as shown in Figure 1. As the second example, "LCD" in ATM system or "LED" in voice mail system (See Figure 2 and Figure 3) have the same concept, other examples, here, in firm planning system are the words "storage" and "inventory". Moreover “year”, “date” and “time” have the same meaning in nearly all different domains.
  - **Note-2:** If different terms are used to describe the same thing in a different semantic domain,

(i.e., to capture a different concept), software engineer needs to capture their concepts with specific details. For instance, "dept" and "loan" may be used by a software engineer as problem domain terms that apply to firm planning system (see Figure 5). However, each term captures a different concept, so these terms represent two different potential objects. Specifically, "dept" related to short-term monetary semantic domain, while "loan" captures a concept in a long-term monetary semantic domain.

- **Note-3:** Sometimes a specific noun is used to capture two different concepts in a single domain. A new term(s) must be created to ensure that each concept/thing is captured. For example, consider the term "cash" in firm planning system. There are two concepts in this domain using this word: (a) we can refer to cash as the money inside the firm, (b) we can refer to cash of the firm in the bank (a part of balance sheet – see Figure 5).
- **Note-4:** Sometimes a specific term is used to capture two different concepts in different domains. The term must be created in a way that each concept/thing is captured. For example, consider the term "account" in ATM system as a class, while it is an attribute of the class USER in the high school system (see Figure 2 and Figure 3).

4. Recommended Approaches

In this section, a couple of approaches are recommended to find objects. In the experiments, we had two groups with different experiences in the object-oriented software development. These approaches, therefore, were based on whether software developers have little or more experience in the object-oriented development.

When software developers have little experience, the following steps are recommended:

1. Given a requirements document of the software in narrative text (English or some languages else) that uses the terms of the domain expert, employ the "Using Nouns" technique. Note that this technique is used to find several potential objects, not all of the objects in the software. In this step, guideline-4 should be considered.
2. Identify all "potential objects" in the problem domain by interactive dialog with the domain experts. Note that we want to capture the objects that are in the mental model of the domain experts. In this step, guideline-4 plays some important roles.
3. Employ the "Using the things to be modeled" technique to elicit more potential objects. In this step, guideline-2 and guideline-4 play some important roles to avoid any mistake.

When software developers have more experience, the following steps are suggested:

4. Underline all of the nouns in the requirements document or use cases. In this step, guideline-1 and guideline-4 should be considered.
5. Filter the list of nouns to identify things outside the scope of the system. These are usually "external objects" or “boundary objects” to which the system interfaces. These external objects will be useful for the context diagram, but it is helpful to keep these objects in the context diagram. Technically, they are not objects in the final model of the systems, so they are not the objects we want to refine. We can then eliminate them from our list of potential objects as part of the systems. In this step, guideline-3 and guideline-4 should be considered.
6. Use the category list given by Coad and Yourdon, Shlaer and Mellor, and Ross to check if there are other concepts or objects that should be added to the list. In this step, guideline-2 plays important roles to avoid any mistakes.

5. Summary and Conclusion

This paper reviewed the techniques to find objects in object-oriented software development and made six taxonomies for them. The techniques covered, here, were Using Nouns, Using Traditional Data Flow Diagrams, Using object-oriented domain analysis, Reusing an Application Framework, Reuse Class Hierarchies, Reuse Individual Objects and Classes, Using Generalization, Using Subclasses, Using Subassemblies, Using Object Decomposition, Using Personal Experience, Using Class-Responsibility-Collaboration Cards, Using the definitions of objects and classes and Using things to be modeled. To get some experience in practice, the techniques were applied to four systems including two system software and two applications. Then, a couple of approaches were recommended for finding objects in the object-oriented development. The approaches obtained would be helpful to develop new systems. Our findings were to use a mixture of the techniques and employed experts to implement and get the best software products in practice. We also classified the techniques as two types, conventional and modern. The modern techniques focus more on discovering a domain model rather than using existing domain knowledge. We feel that it is a good practice for individual learning object-modeling techniques to understand and apply these techniques before using Modern techniques.

TABLE 1
A SUMMARY OF THE TECHNIQUES TO FIND OBJECTS

Taxonomy	Technique	Major Assumptions	Strengths	Weaknesses
1st Document View	UN	Assumes that written documents about the domain exist.	(a) An effective communication medium for both technical and nontechnical project Staff; (b) one-to-one mapping from nouns to objects or classes	It has shortcomings; i.e., It sometimes fails to identify all objects and sometimes identifies false objects
	UDFD	Assumes that a data flow diagram in the domain exist.	(a) Requires no paradigm shift by the analysts and developers; (b) Direct and effective	Requires significant training, practice, intuition, and experience
2nd Knowledge View	UOODA	Assumes that an OODA has already been performed in the same problem domain.	Supports reuse and tends to maximize cohesion in classes and minimizes message and inheritance coupling	Tailoring for performance and other business constraints in a specific project may be lower reuse
	RAF	Assumes that at least one OODA has been done to create an application framework of reusable classes.		Developers must be able to identify one or more relevant application frameworks
	RCH	Assumes that a reuse repository with relevant reusable class hierarchies has been developed.		(a)The existing classification hierarchies may not be relevant to the current application; (b) Existing classes may need to be parameterized, or new subclasses may need to be derived
	RUIOC	Assumes a reuse repository with relevant reusable objects and classes has been developed.	Inexpensive and easy to use	Shortcomings
3rd Commonalities View	UG	Assumes objects are identified prior to their classes.	Promotes reuse and supports the development of one or more classification hierarchies	Requires significant training, practice, intuition, and experience
	USC	Skips finding objects and directly starts identifying classes.	Promotes reusability	When misused, it leads to unmaintainable and opaque classes
4th Decomposition View	USM	Assumes developers are incrementally developing subassemblies using a recursive development process.	(a)Supports incremental identification of objects/classes; (b) Identifies all the subassemblies in an application domain	Identifies only assembled objects
	UOD	Assumes most object are composed of other objects.	May be a better model of the implementation components	Leads to both subtle modeling and technical issues
5th Experience View	UPE	Assumes that the developer has already performed an analysis and can use that experience for this analysis.	(a)Provides a reasonable "reality check" on the current project; (b) Improve the quality of the classes and objects	Developers have a tendency to identify suboptimal classes
	UCRC	It is a human activity that can be stimulated by the use of small pieces of paper.	Inexpensive and easy to use	Developers must have significant experience, creativity, and intuition
6th Abstraction View	UDOC	Assumes that the software engineer has experience in identifying objects and classes.	Direct and effective	Requires significant training, practice, intuition, and experience
	UTBM	Recognizes that the application domain entities need to be identified before identifying the corresponding objects and classes.	Highly effective	Requires significant experience with OO to apply successfully

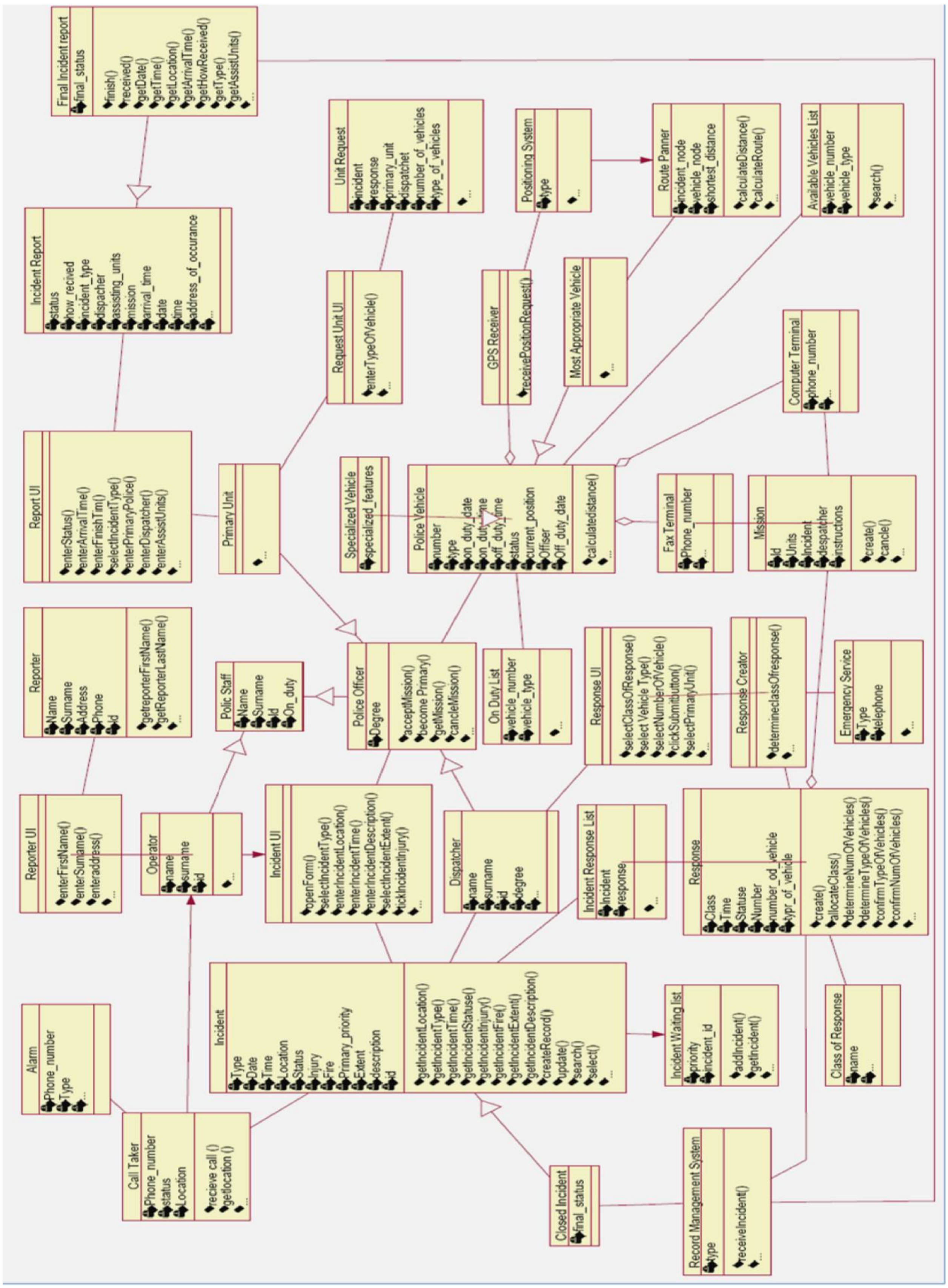


Figure 1: The class diagram of the control command police system[35].

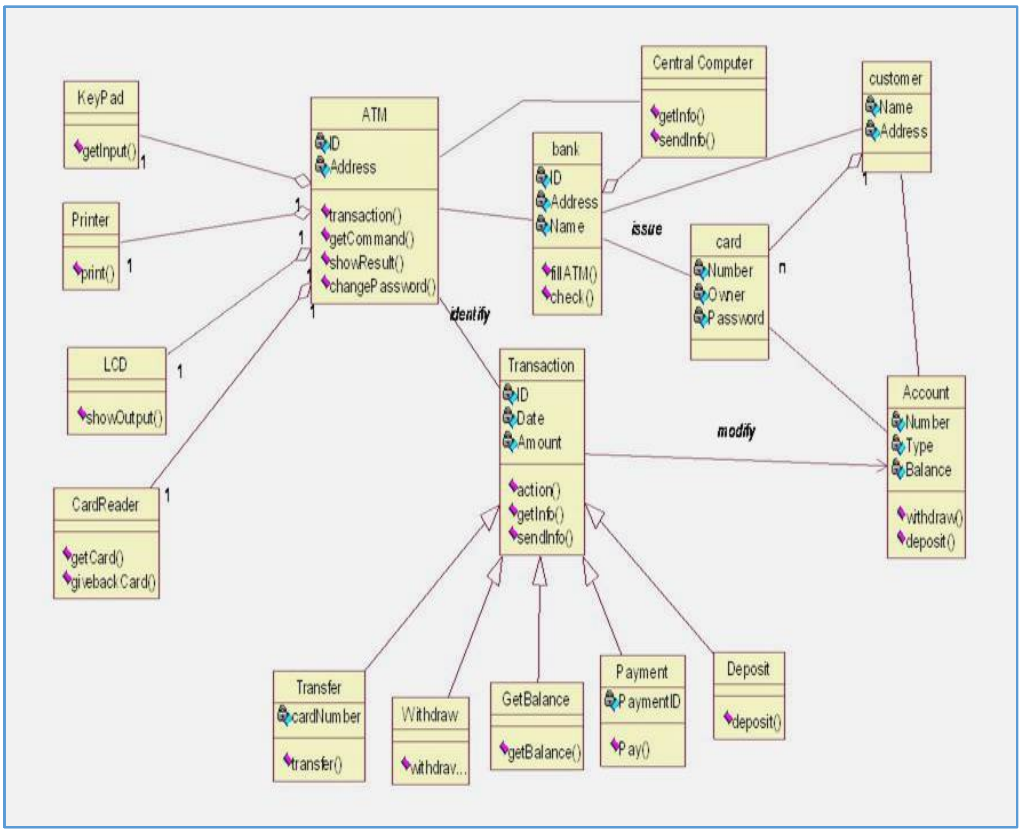


Figure 2: The class diagram of the ATM system.

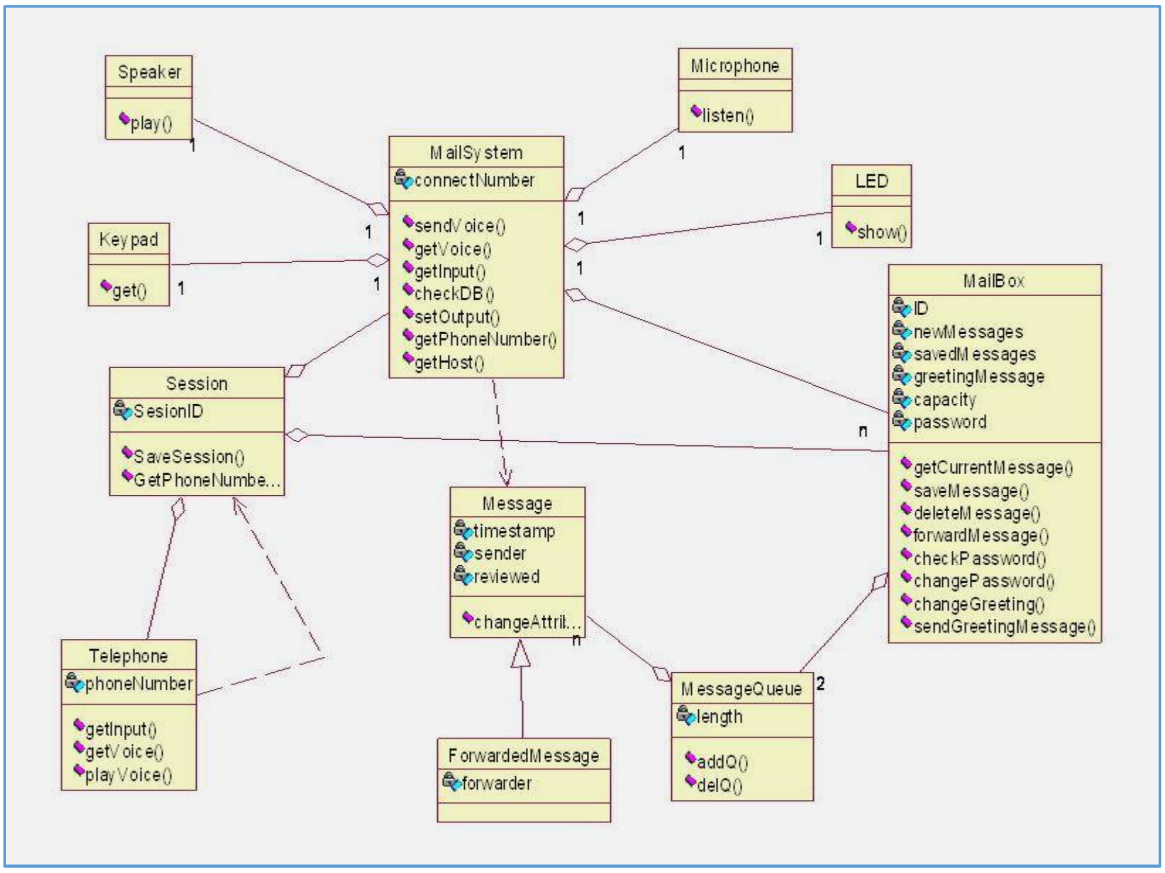


Figure 3: The class diagram of the voice mail system.

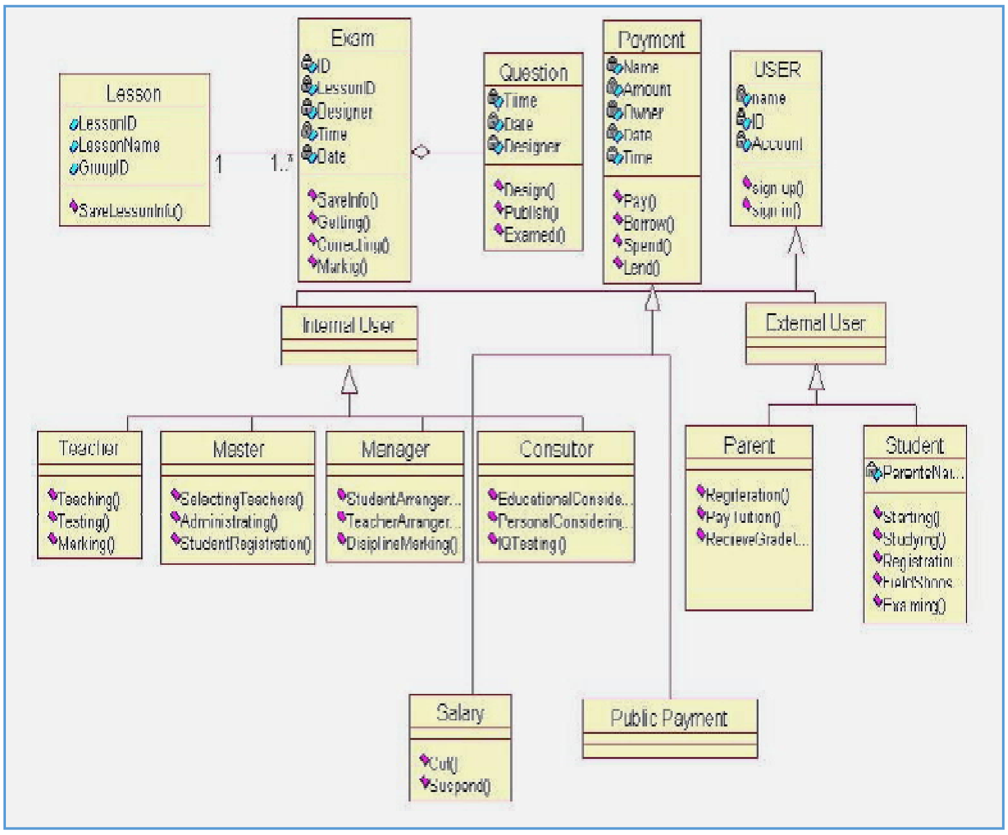


figure 4: The class diagram of the high school system.

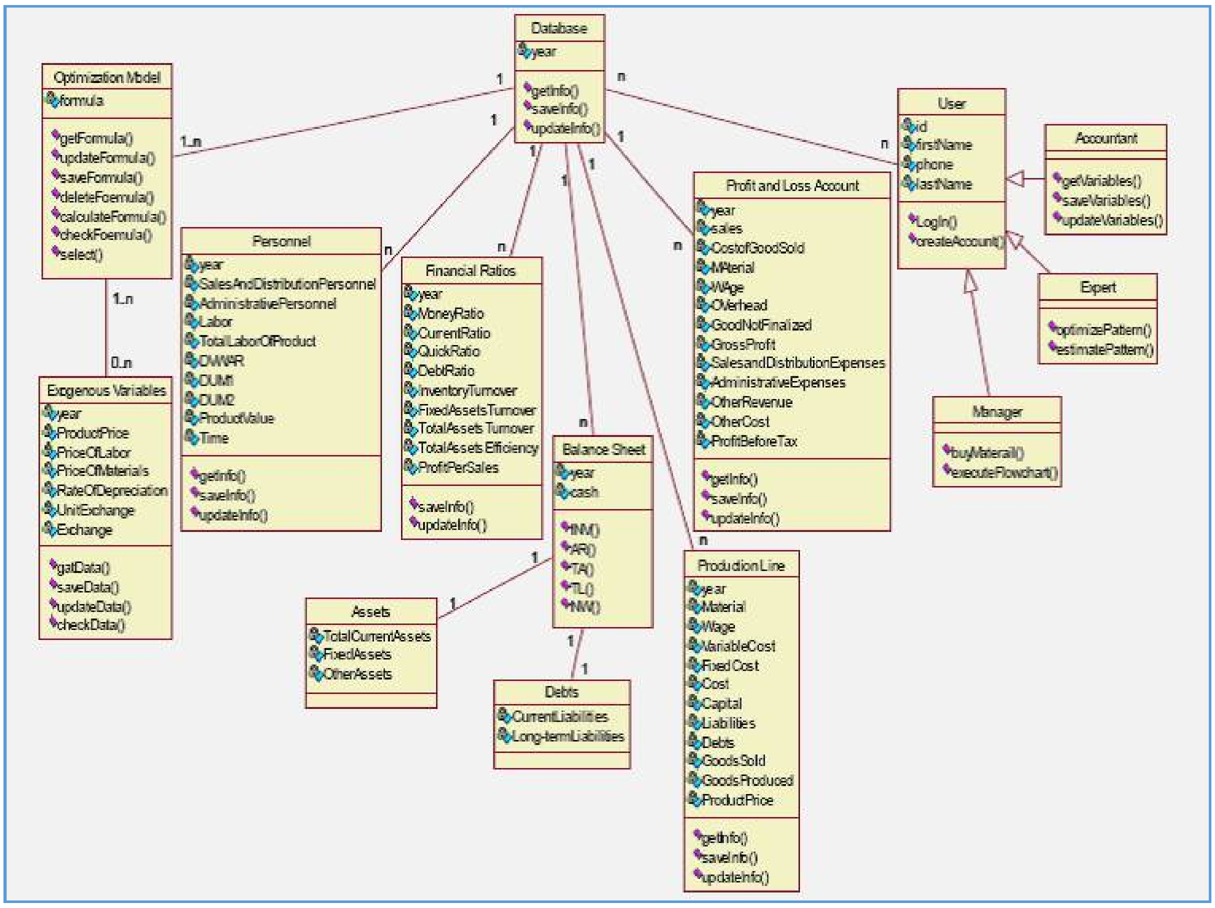


Figure 5: The class diagram of the firm planning system.

TABLE 2
THE AVERAGE NUMBER OF OBJECTS IDENTIFIED BY EACH GROUP

Technique	Systems/Applications															
	High School System (15)				Voicemail System(11)				Firm Planning System (14)				ATM System (16)			
	G_1R_1	G_1R_2	G_2R_1	G_2R_2	G_1R_1	G_1R_2	G_2R_1	G_2R_2	G_1R_1	G_1R_2	G_2R_1	G_2R_2	G_1R_1	G_1R_2	G_2R_1	G_2R_2
UN	9	11	12	15	7	10	10	11	9	12	11	14	12	14	14	16
UDFD	10	11	11	14	8	8	8	10	9	10	10	13	11	12	12	15
UOODA	10	11	11	12	7	8	8	9	9	10	10	11	11	12	12	13
RAF	11	12	12	13	8	9	9	10	10	11	11	12	12	13	13	14
RCH	6	7	7	9	4	5	5	7	6	7	7	8	6	7	7	10
RUIOC	5	6	6	8	4	4	4	6	5	6	6	7	5	6	6	9
UG	5	6	6	8	5	6	5	6	5	6	6	7	5	6	7	9
USC	5	6	6	7	4	5	4	5	5	6	6	7	5	6	6	7
USM	4	5	5	7	3	4	4	5	4	5	5	7	4	5	5	9
UOD	4	5	5	6	3	4	4	4	4	5	5	6	4	5	5	6
UPE	10	14	13	15	7	10	10	11	8	12	11	14	11	15	15	16
UCRC	10	11	11	14	7	8	8	10	9	10	10	13	11	12	12	15
UDOC	6	7	7	8	4	5	5	6	6	7	8	10	6	7	7	9
UTBM	11	12	12	13	8	9	9	10	10	11	11	12	12	13	13	14

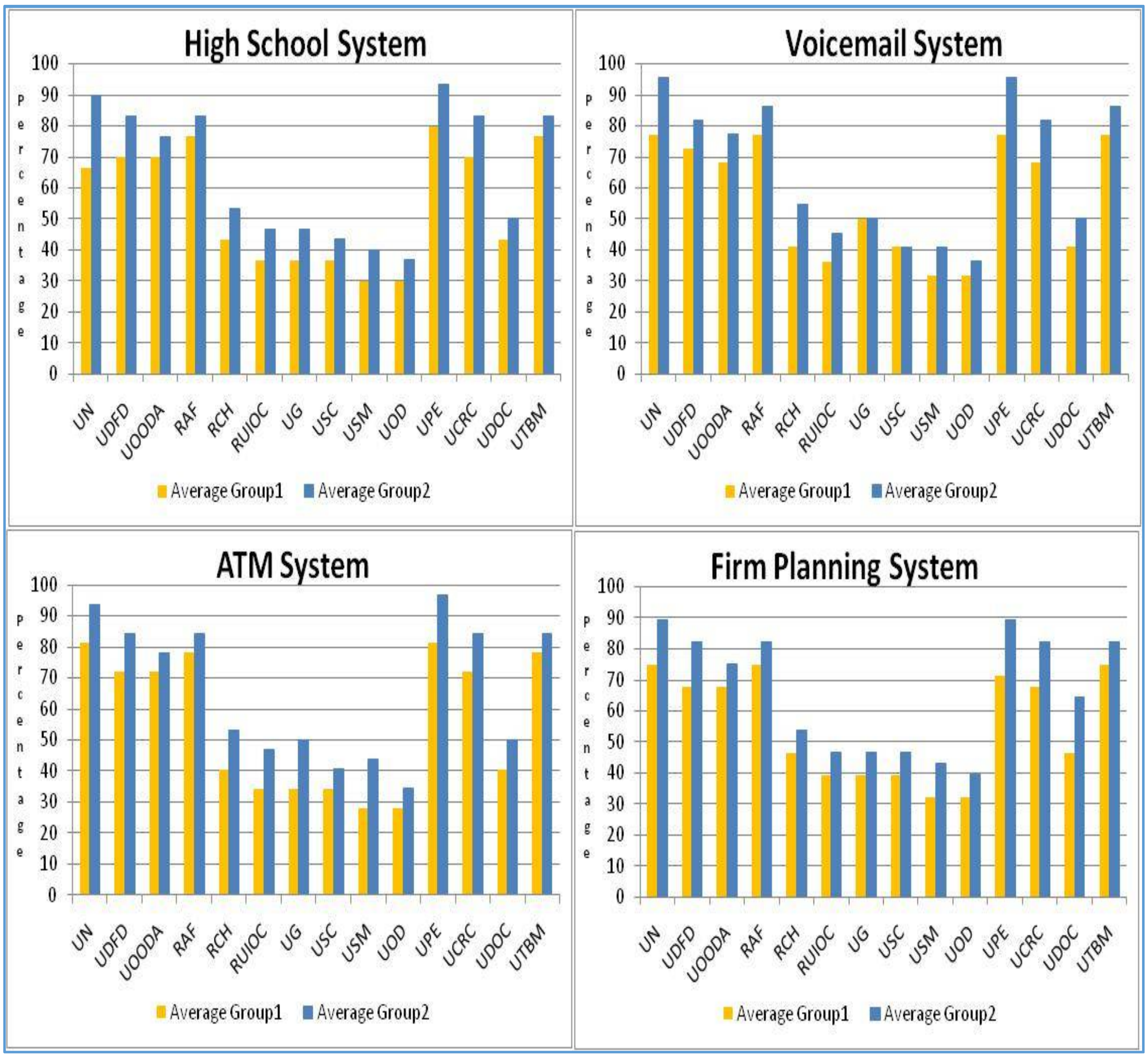


Figure 6: Average percentages of the objects identified by applying the techniques.

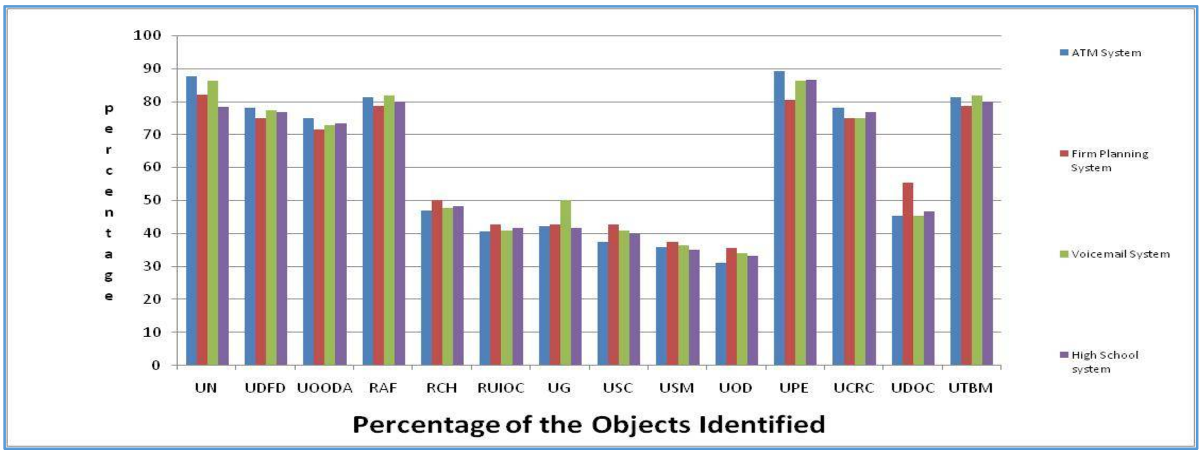


Figure 7: Average percentages of the objects identified in the four systems by applying the techniques.

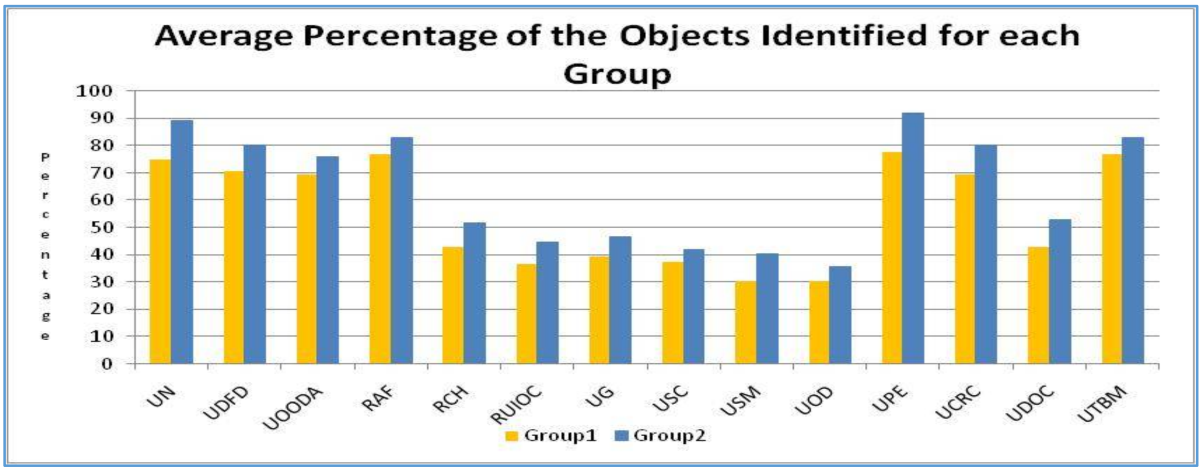


Figure 8: Average percentages of the objects identified by applying the techniques in each group of participants.

TABLE 3
POSSIBLE CASES FOR USING TERMS

	Concepts	Domains	Note
Terms (Nouns/ Words)	Same	Different	1
	Same	Same	2
	Different	Same	3
	Different	Different	4

REFERENCES

[1] G. Bavota, A D. 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.

[2] G. Nanda, N. C. Kar, “A Survey And Comparison Of Characteristics Of Motor Drives Used In Electric Vehicles,” *IEEE Electrical and Computer Engineering Conf*, pp. 811-814.

[3] G. Booch., “Object-Oriented Development,” *IEEE Transaction on Software Engineering*, 12 (2), pp. 211-221, 1986

[4] G. Booch., “Software Engineering with Ada,” *Benjamin/Cummings Publishing Co.*, Menlo Park, California, 1983

[5] G. Booch, J. Rumbaugh, and I. Jacobson, “The Unified Modeling Language User Guide,” Addison Wesley, 1998

[6] G. Booch, J. Rumbaugh, and I. Jacobson “The Unified Software Development Process,”Addison-Wesley, 1998

[7] F. P. Brooks, “The Silver Bullet, Essence and Accidents of Software Engineering,” *Information Processing* '86. Ed., Kugler H. J., Elsevier Science Publishers B.B. (North-Holland), 1986

[8] F. P. Brooks, “The Mythical Man-month: Essay on Software Engineering,” Addison-Wesley, 1982

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

[10] 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

[11] Ch. Peter, “The entity-relationship model-Toward a unified view of data,” ACM Trans. on Database Systems, Vol. 1(1.), pp. 9-36, 1976

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

[13] A. Cockburn., “Writing Effective Use Cases (Draft 3),” Addison Wesley Longman, 2000

[14] E. Codd, “Extending the database relational model to capture more meaning,” *ACM Trans. on Database Systems*, Vol. 4(4), pp. 397-434, 1979

[15] A. V. Deursen, 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

[16] Faculty of Electrical, Computer and IT engineering, Islamic Azad University, Qazvin Branch, <http://qiau.ac.ir/en/faculties/counter/dep.aspx?d=0>

[17] 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.

[18] M. Fowler, , and K. Scott, “UML Distilled A Brief Guide to The Standard Object Modeling Guide,” 2ndEdition,Addison Wesley Longman, Inc, 1999

[19] N. Goldsein, and J. Alger “Developing Object-Oriented Software for the Macintosh Anaiysis, Design, and Programming,” Addison-Wesley, 1992

[20] 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

[21] I. Jacobson. and G. Booch, “The Unified Software Development Process,” Addison-Wesley, Reading, MA, 1999

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

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

[24] R. King, “My Cat Is Object-Oriented”, Object-Oriented Concepts, Databases and Applications”, Addison Wesley, 1989

[25] M. Langer, “Analysis and Design of Information Systems”, 3rdEdition, Springer-Verlag London Limited, 2008

[26] R.C. Lee and W.M. Tepfenhart, “UML and C++: A Practical Guide to Object-Oriented Development,” 2ndEdition, Pearson Prentice Hall, 2005

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

[28] S. M. McMennin, and J. F. Palmer. Essential System Analysis, Yourdon Press, 1984

[29] Merriam-Webster Online (2011), Dictionary and Thesaurus, fromhttp:// www.merriam-webster.com

[30] B. Meyer, “Object-Oriented Software Construction,” Prentice-Hall International (UK) Ltd., Cambridge, UK, 1988

[31] Musser, R. David, and A. Saini., “STL Tutorial and Reference Guide C++ Programming with the Standard Template Library,” Addison Wesley, 1996

[32] S.h. Pfleeger, and J.M. Atlee, “Software Engineering: Theory and Practice,” 4th Edition, Pearson, 2010

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

[34] M. R. Quillian, “Semantic Memory In Marvin Minsky,” *Semantic Information Processing*. Cambridge, MIT Press, 1968

[35] H. Rashidi, “Software Engineering-A programming approach,” *2ndEdition, AllamehTabataba'i University Press* (in Persian), Iran, 2014

[36] D. Ross, “Applications and Extensions of SADT,” *IEEE Computer*, 1985, Vol. 18 (4), pp. 25-34.

[37] J. Rumbaugh, “Getting Started: Using Use Cases To Capture Requirements,” *Object-Oriented Programming*, Vol. 7(5), pp. 8-12, 1994

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

[39] S. Schlaer, and S. Melior, “Object-Oriented Systems Analysis: Modeling the World in Data,” *Yourdon Press*, 1988

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

[41] Y. Sommerville, “Software Engineering,” 9th Edition, Pearson Education, 2010.

[42] L. A. Stein, , H. Lieberman, and D.Ungar, “A shared view of sharing: The Treaty of Orlando,” Object-Oriented Concepts, Databases, and Applications”,Eds. by W. Kim , and F. H. Lechosky, ACM Press, New York, 1989.

[43] B. Stroustroup, “The C++ Programming Language,” Addison-Wesley, 1991

[44] K.S. Subhash et al., “NLP based Object-Oriented Analysis and Design from Requirement Specification,” *International Journal of Computer Applications*, , Vol. 47 (21), 2012

[45] M. E. Winston, R. Chaffer, and D. Herrmann, “A Taxonomy of Part-Whole Relations,” *Cognitive Science*, Vol. 11, pp. 417-444, 1987.

[46] R. Wirfs-Brock, “Designing Object-Oriented Software,” Prentice-Hall, 1990

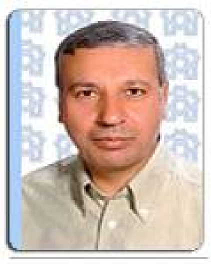
[47] E. N. Yourdon, and L. L. Constatine,“Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design,”Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

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

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

[50] H. Rashidi, “A Systematic Approach to Financial Planning in Firms and Its Implementation in an Enterprise,” Quarterly Journal of Fiscal and Economic Policies, Vol. 2 (8), PP. 73-92, 2014.

BIOGRAPHIES



Hassan Rashidi is an Associate Professor in Department of Mathematics and Computer Science of Allameh Tabataba'i University. He received the B.Sc. degree in Computer Engineering and M.Sc. degree in Systems Engineering and Planning, both from the Isfahan University of Technology, Iran. He obtained Ph.D. from Computer Science and Electronic System Engineering department of University of Essex, UK. His research interests include software engineering, software testing, and scheduling algorithms. He has published many research papers in International conferences and Journals.