Seven Steps for Object-oriented Normalization in
Class Diagrams: Example of Jigsaw Puzzle Concept
for Image Retrieval

Sheng-Hung Lo*, Yih-Chearng Shiue and Kuan-Fu Liu

Department of Information Management, National Central University,
Taoyuan, Taiwan 320, R.O.C.

Abstract

Despite the growing popularity of object-oriented programming in recent years, a comprehensive
discourse for object-oriented normalization has been lacking. First, the present study investigates the
theoretical basis of object-oriented normalization from the perspectives of encapsulation, inheritance,
and polymorphism, which were considered the three main features of object-oriented programming.
This study further identifies the logic and rules for object-oriented normalization and translates them
into seven steps for applying object-oriented modeling in class diagrams, generating normalized
databases, and establishing object-oriented operating structures. Second, the use of the formalization
of the object-oriented jigsaw puzzle concept in the implementation of an image search model enhances
the artificial intelligence for the retrieval speed for a large amount of data.

Key Words: Object-oriented, Normalization, Class Diagram, Jigsaw Puzzle Concept, Image
Retrieval, Artificial Intelligence

1. Introduction

Since the emergence of object-oriented programming, the information technology and related industries have
employed it in the development and design of systems and process concepts to such an extent that it was worth
reorganizing and rethinking its logic and rules to resolve the problems in system development posed by normal-
ization. In object-oriented system design, the consistency, completeness, and accuracy of class diagrams determines
the success (or lack thereof) of a system [1,2]. Clear and specific principles and procedures have been established
for conventional structured system design through the use of entity relationship diagrams and database normal-
ization. However, in regard to object-oriented system design, although shared operation as the object-oriented
principle for the second normal form has been proposed, there are still no specific rules for the first, third, and
fourth normal forms, nor are there considerations for object-orientedness. Therefore, the present study investi-
gated the theoretical basis of database normalization from object-oriented programming’s characteristics of encaps-
ulation, inheritance, and polymorphism to propose principles for the first to fourth normal forms and to translate
such principles into specific modeling procedures in class diagrams for the purpose of generating a normalized da-
tabase and operable object classes.

Through object-oriented technology, the characteristics of the picture such as texture, color, and shape are
encoded into the database package as an object. The purpose of this paper was to improve the image search sys-
tem accuracy and speed of comparison [3]. This study’s presented detection technology was expected to identify
significant object images through the significant items

*Corresponding author. E-mail: shenghung.lo@gmail.com
This paper is an extension of the authors’ technical abstract presented in the 6th International Conference on Frontier Computing (FC 2017)
held by The Institution of Engineering and Technology in Osaka, Japan from July 12-14, 2017.
and reduce the influence of background noise on the object. The significant object detection processed image was used in scale-invariant feature transform (SIFT) [4] features to capture an image. Then, through the K-mean clustering algorithm, feature vectors for all images were clustered to obtain a package of images features (bag of features, BOF) [1,5–7] vectors as the basic conditions for image retrieval. This system was expected to improve the accuracy of image searching and the image searching system through the object-oriented search pattern (such as the jigsaw puzzle, we call it jigsaw puzzle concept (JPC)) to attempt to solve the huge amount of data [8,9].

2. Literature Review

2.1 Class Diagrams

The class diagram is a type of static structure diagram in software engineering. It uses the unified modeling language (UML) to describe a system’s classes, their attributes, the structure of operations, and the interclass relationships [2,10]. Classes constitute the core of an object-oriented system since they help translate the model into scripts [1,11,12].

Various types of relationships can exist between classes (either hierarchical or parallel), each of which can be denoted by a distinct symbol [13,14]. The multiplicity in the relationships must also be considered. Depending on the number of objects involved, multiplicity values must be assigned to signify the number of instances of the associated classes, which can be one-to-one, one-to-many, or many-to-many.

2.2 Object-oriented Programming

Object-oriented system development has the primary task of identifying the relationships between the data and operations to solve problems through the collaboration of said objects. The concept of object-oriented programming enhances the repeatability of program components and views the objects as basic elements of a program that encapsulate sets of attributes and operations to enhance flexibility and expandability. Object-oriented programming has three primary features: encapsulation, inheritance, and polymorphism.

Encapsulation is a deliberate design choice to hide the specific data and functions of an object and, in their place, use message-passing mechanisms to enable inter-object communication and collaboration. This design establishes a restriction that only objects of a certain class can access members of this class, such that the communication between classes is done through passing messages or invoking methods. The purpose of encapsulation is to allow the objects to bypass the details in interobject communications and concentrate on their own functions, thus reducing the dependency between classes.

Inheritance refers to the forming of the parent class and its subclasses. The shared features between classes (such as attributes and functions) can be generalized and reorganized to form a parent class (or super class). Those features are then inherited by subclasses that, aside from the attributes and functions inherited from the parent class, are also given their own attributes and functions to make them even more specific [15]. The purpose of inheritance is to recycle or inherit existing code for the attributes and functions shared between classes to save time in coding the system.

Polymorphism refers to varied reactions to the same message as a result of multiple subclasses that inherit the same parent class. When various classes inherit the functions of the same parent class, they may be called to perform a function that bears the same name in the subclasses but returns a variety of outcomes because of the diverging needs of the distinct subtypes. A feature called “override” is thus derived from polymorphism. It is an exception made for a subclass inheriting a function from the parent class to perform a specialized function or produce a specialized outcome. In other words, override involves a subclass redefining the function inherited from its parent class to suit the needs of its class [16].

2.3 Database Normalization

For relational databases, normalization is an essential principle and technique that facilitates the systematic organization of data and the elimination of repeated content. It reduces redundant information in the database and improves the cohesion of the schema, thereby ensuring the consistency, completeness, and accuracy of the data [17–22].

The first database normal form (1NF) is introduced to eliminate repeating groups in the data. It works by requiring that each attribute contains only one value. That
is, composite data and multivalued data are eliminated to conform to the first database normal form. The second database normal form (2NF) refers to a database that meets the criteria for the 1NF; furthermore, all of its data are required to be absolutely dependent on the table’s primary key. Any partial dependency among the data is eliminated. The third database normal form (3NF) requires the table to be in the 2NF; to ensure that all non-key attributes are absolutely dependent on the primary key, no interdependence can exist between the non-key attributes. In other words, any transitive dependency between attributes is eliminated. The fourth database normal form (4NF) requires conformity to the 3NF, and no combination in the table can be a repetition of the primary key [7]. In other words, multivalued dependency on the primary key must be eliminated.

2.4 Object Normalization

In object-oriented system design, the data and functions of the objects require careful organization to reduce the dependency between the objects and enhance the cohesion between them. Therefore, the schema of an object-oriented database must also be normalized by reorganizing the attributes of the objects and the structure of the functions in order to establish an adequate classification of the objects [23,24]. For a class to be in the first object normal form (1ONF), it must be in 1NF to remove any repeating groups of composite or multivalued attributes, and it also has to avoid functional overlaps in its operations [24,25]. This means that composite operations are eliminated. A class in second object normal form (2ONF) must be in 1ONF, and the partial dependency attributes must be isolated; shared operations should not exist in the class [24]. To be in third object normal form (3ONF), a class must be in 2ONF and the transitive dependency between the attributes is eliminated. Shared operations must also be normalized through homogenous operations to maintain the flexibility between varied classes [26]. A class in fourth object normal form (4ONF) is in 3ONF, and no multivalued dependency exists in the combinations of its set of determinants when the class determinant consists of multiple determinants. The operations in the class also do not access the multivalued dependent determinant, which means that all multivalued dependency is eliminated from the operations.

2.5 Salient Object Detection in Images

Before the human eye engages in image recognition, it identifies the salient regions of an image with remarkable precision and speed. Therefore, approximating human object recognition capabilities is a major topic in machine vision and has received extensive attention. In this report, we adopted the salient object detection method proposed by Liu, Sun, Zheng, Tang, & Shum [5]. They proposed a supervised approach to salient object detection that used a set of novel features to describe salient objects and optimally combined three methods through conditional random field learning. These methods are the center-surround histogram (CSH), multiscale contrast (MSC), and the color spatial distribution (CSD). Through the way that the feature map is normalized, these three methods are optimally combined through conditional random field learning to generate a binary picture that separates a salient object from the background. These visual attention studies concluded that contrast is the most common parameter in detecting local features because contrast operators model the human visual receptive fields [27–29]. For this reason, when the size and location of a salient object are unknown, contrast was typically estimated at multiple scales, as shown in Figure 1.

3. Object-oriented Features and Object Normalization

3.1 Object-oriented Framework

The present study primarily relies on analyzing the logic behind database normalization, the theoretical ba-
sis of object normalization, and object-oriented design techniques and characteristics to propose logic rules for the normalization of object-oriented design. The core of the research framework has been to propose a whole new set of methods and procedures for the normalization of class diagrams, as shown in Figure 2.

3.2 Dependency between the Determinants, Attributes, and Operations

The database normalization method proposed by Wu [30] was adapted for class normalization. Because the structure of the system as a whole is determined by the dependency between system attributes and operations, the establishment of dependency was recognized as the core procedure for object normalization.

3.2.1 Attribute-dependent Determinants

The master/slave relationship between system attributes can be learned from the descriptions of the attributes. If A represents the set of system attributes and D represents the set of system determinants, D \subseteq A, and if N represents the nondeterminants dependent on D, N \subseteq A. Furthermore, when A_x \subseteq A and A_y \subseteq A - A_x, the change in an attribute A_x determines the change in another attribute A_y and identifies the change in A_x\rightarrow A_y. This attribute can be named the determinant D_x and the dependent A_y named N_x, where D_x \subseteq D and N_x \subseteq N. When a D_x can be used to determine N_x, it can be referred to as the attribute dependent on the determinant D_x \rightarrow N_x.

Therefore, system attributes could be divided from a master/slave viewpoint into the determinants (Ds) and nondeterminants (Ns). Attributes that were found to be neither D nor N were coded L, and L = A - D - N, meaning that the Ls are not dependent on either of the attributes, and they cannot determine other attributes [30].

Per the preceding description, the first step of normalization should be the division of the attributes into three groups: the determinants, the attributes dependent on the determinants, and the leftovers.

So far, only the situation of a single attribute becoming a determinant has been considered, but a determinant could also be a set comprised of multiple attributes. Moreover, even though an L might not be dependent on a single-attribute determinant, it could in fact be dependent on a multiple-attribute determinant. The nature of a multiple-attribute determinant would involve partial dependency between attributes. That is, N_x \subseteq N and D_x \subseteq D, where the attribute N_x is dependent on the subset of a multiple-attribute determinant D_x' such that N_x \rightarrow D_x \land N_x \rightarrow D_x'. Therefore, of the total number of determinants D_n, the known determinants would be combined and permuted as D_y \subseteq D and L_y \subseteq L. From the smallest set, the determinant D_y would be brought out to be matched from C^{D_n} to C^{D_n-1} to verify whether L_y \rightarrow D_y. Whenever L_y \rightarrow D_y' was found, it was ignored to eliminate partial dependency. That is, only if L_y \rightarrow D_y would the dependent multiple-attribute determinants be added into the set D and the L_y that satisfies the dependency added into the set N.

After all of the determinant combinations were identified and assigned to D, the absolute dependency of the nondeterminants could be determined. L, which includes the attributes that were not related to or dependent on other attributes, was set as a new determinant and subjected to the aforementioned procedures to identify the attributes and operations that were dependent on it until all of the attributes could be classified. Through this method, all of the determinant sets in the system could be identified, and all of the nondeterminant sets were dependent on the determinant sets. The dependencies between the nondeterminants were eliminated after this reorganization of the nondeterminants, thus eliminating the transitive dependency between the attributes [31].

3.2.2 Operation-dependent Determinants

The dependency on a determinant is established th-
through the identification of dependent attributes, whereas the dependency of operations is established through access to the determinant. Unlike the attributes of an operation that are readily known, the determinant can only be defined through the dependency between attributes. The fact that an operation cannot define a determinant means that the process of system modeling has to first define the determinants based on interattribute dependencies and then determine the operations’ dependencies on the determinants. If $O$ represents the set of system operations and $O_x \subseteq O$ if $O_x$ can only access the determinant $D_x$ and the nondeterminant $N_x$ that is dependent on $D_x$ (i.e., $O_x$ cannot access any other attributes that are not dependent on $D_x$), the determinant is referred to as operation-dependent determinant $D_x$. Through the classification process addressed in section 3.2.1, every time a new determinant is found, the operations dependent on it can also be found from the corresponding dependent attributes.

Through this classification process, each determinant corresponds to multiple operations, whereas each operation only corresponds to one determinant. When examining the set of operations dependent on the same determinant, if functional overlapping is found among the operations, the overlapping part would be set aside as a new operation so that the original operation would no longer include the function of the new operation. However, not all of the operations are dependent on one determinant, and if the range of an operation’s access is found to exceed $D_x + N_x$, this means that the range exceeds the access to a class, and further action is required.

### 3.3 Class Generalization and Specialization

After establishing the dependency between attributes and operations, the system’s preliminary data and operational structures are ready for further refinement based on the system’s needs. Generalization involves extracting the common parts of disparate classes as a basic class that is a level higher than other classes. It corresponds with the concept of inheritance in object-oriented programming whose purpose is to simplify the system’s structure while still maintaining the design intent of the original structure. However, specialization forcefully breaks up the existing logic and rules of the class structure through processes or outputs that are unique in use case diagrams. It fulfills the special needs of the exceptional cases in object-oriented programming with minimal overriding of existing scripts.

#### 3.3.1 Homogeneous Operations and Inheritance

If the classes generated in section 3.2 were subjected to tests for attributes and operations of the same functions, this would probably generate so many combinations of attributes and operations with a common structure that an adequate common structure would be impossible to define. A homogeneous operation refers to the situation where attribute sets $A_x$ and $A_y$ from separate classes have the same function and are accessed in their respective classes by operations $O_x$ and $O_y$, both of which also have an identical function. This means that the attributes and parameters accessed in the homogeneous operation are also indicative of the structure of the attribute sets in their classes, which can be referenced when the common structure between classes has to be established. However, in light of the multitude of single-attribute identical functions in the classes, establishing the common structure based on a homogeneous operation accessing a single attribute would generate too many parent classes with a single attribute or operation, and this would only serve to diffuse the original dependency. To prevent such meaningless increases in classes through class generalization, the present study only conducted generalization on homogeneous operations that accessed more than two attributes in classes with more than two homogeneous operations.

#### 3.3.2 Homogeneous Operations and Polymorphic Operations

The method of using a homogeneous operation to determine the common structure of various classes and attributes (which is then extracted to form a new parent class) can be used to make deductions from existing repeatable structures without influencing the essence of corresponding attributes and operations. This way, multiple subclasses inherit the homogeneous operation of the same parent class to complete the procedures of their own classes, which conforms to the polymorphic feature of object-oriented programming. Furthermore, the common inherited structure also helps enhance the flexibility of the system, thus making it conform to the 3NF.

Because the homogeneous operation uses the attrib-
utes and parameters accessed for the operation as the reference for generalization, the inheritance of operations in the parent class using the polymorphic characteristics does not take the needs of the subclass into account. Hence, in the exceptional case where inherited polymorphic operations do not meet the needs of the subclass, overriding could be used as a specialization method to achieve adaptation with minimal change. The actual needs can be determined by examining the procedures and expected outputs of the operation, which would then be compared with the application of homogeneous operation inherited from the parent class to determine whether overriding would be required.

3.4 Class Encapsulation

After the adjustments in section 3.3, the inheritance of different classes has been determined though the homogeneous operation, and the class attributes and operations can be encapsulated into official classes. The encapsulation of a class can be realized through its visibility. A concrete class must come with a name and include the attributes and operations of the association (which must fit to a predetermined visibility) and the actual data type [23].

To enable the ability to call the operation from other classes, the visibility of the operation should be set as “public”. The visibility of an attribute should be determined by the access granted to the system operation. Conversely, to protect the integrity of the encapsulation, the corresponding visibility should be set as “private”. When an attribute is accessed by an operation that is not in an attribute-dependent determinant (i.e., the operation needs to access the attribute across class boundaries), it should be done by calling the operational attribute of the class. When the attribute is in a parent class of other classes and the subclass has inherited it, this means an operation that is inherited from that parent class is calling on that attribute, and thus, the visibility of the attribute should be set as “protected” to allow access for that subclass.

3.5 Control Classes

Given the types of dependencies discussed in section 3.2, examining whether the range of an attribute accessed by an operation is wide enough to encompass a determinant set and its dependent nondeterminants, and then, adding the operation dependent on the determinant set into a concrete class in the fourth step can leave some leftover operations (Ls) that have been untreated since the second step. These Ls could be dependent on any determinant (D), and the range that these operations access does not follow the dependent attributes, thus placing it beyond the attribute set that is dependent on one single determinant. Moreover, although the dependency on a single determinant cannot be determined for the class operation (thus disqualifying it for a concrete class), this also means that the access authority of this type of operation is higher than ordinary operations of concrete classes. An operation of this type is able to control attributes in multiple classes, which conforms to the shared operation of 2ONF, and should be classified under the control class.

3.6 Multiplicity in Interclass Relationships

After establishing the concrete and control classes, the sixth step is to define the relationship between the classes. A notation could be used to indicate the hierarchy and active/passive statuses in such relationships. A class could maintain a relationship with multiple other classes, each of which exhibits multiplicity in the relationship. The relationship corresponding to a single-determinant class can vary considerably from one to a multiple-determinant class; the former can only be inferred from use case diagrams and logic, whereas the latter can be inferred from the set of class determinants. A multiple-determinant class is a concrete class made up of the relationship between two classes. Each sub-determinant in a multiple-determinant class corresponds to the relationship with one other single-determinant class [13, 14, 30].

Therefore, first the relationships with single-determinant classes should be established, and then, a multiple-determinant class can be inserted between two single-determinant classes. Because the control classes do not have a determinant, they can access the concrete class attributes of the aforementioned single-determinant classes and multiple-determinant classes. The relationship where control classes use concrete classes and the used concrete classes influence the control classes is a temporary weak relationship; each time an attribute in a concrete class is accessed, a “use” relationship would be established with that class.
When the class determinant set is composed of more than three determinants, the class is in multiple relationships with more than two other classes. Because such a class does not conform to 4NF multivalued dependency, the class determinant has to be dismantled into a set of two or fewer determinants, which should still maintain the multiplicity of the corresponding relationships. However, the multiplicity in relationships can lead to dependency problems. In the following sections, the study discusses the multiplicity problems of one-to-one and many-to-many relationships according to the records of various classes that were paired by determinants. When there is a multivalued dependent determinant in a class, there are also operations dependent on it. Therefore, when eliminating multivalued dependencies, the operations accessing the multivalued dependent determinant must also be examined for adjustments. When a new attribute is assigned to a class, the class operations must be adjusted as a result. Likewise, if the adjusted determinant forms a multiple-attribute determinant with the original class determinant, all the operations accessing the determinant must be adjusted accordingly. Conversely, if the adjusted determinant serves only as a class nondeterminant instead of forming a multiple-attribute determinant with the original class determinant, and the original class operations do not access it either, then only the operations accessing the determinant have to be altered in response to the change.

3.6.1 Determinants Adjusted by One-to-one Multiplicity

In a one-to-one relationship, a class corresponds to only one other class, with either of them being the single-attribute determinant class. Because the single-attribute determinant class cannot record the relationship with other classes in the system logic, adjustments have to be made through the class primary key. To reflect the corresponding relationship in the system, the determinant of one of the classes is assigned to another. The changed determinant cannot be combined with the original determinant of the class as a multiple-attribute determinant unless it leads to the partial dependency of the multiple-attribute determinant set that is not in 2NF. Therefore, the changed determinant should be independent of the other nondeterminants and serve only as an independent non-determinant that is in 3NF. When the class relationships on both sides conform completely to the criteria, thus keeping the changes minimal, the determinant from the class with more operations is reassigned to the attributes of the other class to adjust the operation parameters of the target class. With the addition of the new determinant, class operations have to be adjusted one by one to make sure that the determinant only acts as a class nondeterminant. Only the operations in the class that access the determinant are changed to add the adjusted determinant into the attributes and parameters that access the determinant.

3.6.2 Determinants Adjusted by One-to-many Multiplicity

In a one-to-many relationship, a class can correspond to multiple other classes, which in turn correspond to a single class. In the event that the multiple other classes are single-determinant classes, by adjusting the multiple classes to record how the classes correspond to other classes, the determinant of a class can be inserted into the multiple-attribute compartment. The determinant of the multiple classes generates multiple combinations. Therefore, the changed determinant and the original class determinant should be combined as a multiple-attribute determinant set. When a determinant is assigned to multiple classes, the operations of the accessing class determinant should be adjusted accordingly. Then, the changed determinant is combined with the original class determinant to form a multiple-attribute determinant. The operations corresponding to the original class determinant should also be changed to incorporate the changed determinant into the attributes and parameters so that the class operations remain dependent on the new determinant set.

3.6.3 Determinants Adjusted by Many-to-many Multiplicity

In a many-to-many class relationship, numerous corresponding relationships with other classes are found on both ends. Because the multiplicity of relationships cannot be determined for the determinants of both classes, to record the relationship between the classes, the determinants of the two classes are placed into a new class. The new class must follow the dependency relationship of the attributes and operations. Attributes dependent on
the class determinants that are used to form the new class determinant set have to be moved from the multiple classes to the new class. Likewise, operations that only access the moved attributes and are dependent on the new determinant should be moved to the new class.

Following the three preceding multiplicity adjustments, when a determinant is changed, only classes with a single attribute can remain. After the logic adjustments, every class can exhibit a relational multiplicity in that the single-attribute class exhibits the one-to-many relationship with other classes and the classes corresponding to the single-attribute class contain the determinant for the single-attribute class. The accessed class contains the determinant of the single-attribute class. To reduce inter-class dependency, the fourth rule was established to eliminate single-attribute classes.

3.7 Seven Steps for Object Normalization

Before object normalization, information on the object’s class must be obtained to grasp the logical relationship between the attributes. This information includes names and descriptions of attributes, data types, and examples (Table 1). The required operational information includes the names, descriptions, and parameters of the operations and the accessed attributes (Table 2). The information addressed here must be ready before object normalization can proceed.

As discussed in preceding sections, the object normalization procedure has seven steps, described as follows.

Step 1. Classification of Attributes and Operations
1. Elimination of redundant attributes and composite attributes: As shown in Table 3, attributes are classified as determinants (A1), dependent on determinants (A2), or other (A3), whereas operations are classified as dependent on determinants (O1) or other (O2).

2. Elimination of composite operations: This stage looks for overlaps between O1 operations that are dependent on the same determinant. The overlap is made into a new operation.

Step 2. Reorganization of Multiple-Dependent Attributes and Operations
1. Elimination of partial dependent attributes and shared operations: This stage determines the attributes and operations in A3 and O2 that can be decided concurrently and by which determinants.

2. Elimination of transitive dependency: Attributes in A3 that are not dependent on any determinants are redefined as new determinants and then added to A1.

Step 3. Encapsulation of Inherited Classes
1. Elimination of homogeneous operations: If two classes have attributes of the same function or two or more operations that access such attributes, these attributes and operations should be removed from the original classes and encapsulated as a new parent class.

2. If there are two subclasses that inherit the polymorphic operations of a parent class, whether either of the subclass operations needs to override the parent class operation is determined depending on the polymorphic operation of the parent class.

Step 4. Encapsulation of Concrete Classes
1. Individual concrete classes are encapsulated according to the determinants and named. The visibility and data type are clearly marked for each attribute, and visibility is also clearly marked for each operation.

Step 5. Encapsulation of Control Classes
1. Operations in O2 that access attributes across classes are encapsulated as control classes and named. Visibility is also clearly marked for each operation.

<table>
<thead>
<tr>
<th>Table 1. List of class diagram attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2. List of class diagram operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3. Attributes and operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determinant attribute dependent on determinant (A1)</td>
</tr>
<tr>
<td>---------------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Step 6. Plotting the Class Diagram
1. The relationships between concrete classes are plotted based on the logical relationships derived from use cases.
2. The relationships between control classes are plotted based on the accessed attributes of concrete classes.

Step 7. Adjustments for Class Relationships
1. If two classes are in a one-to-one relationship, the determinant of the class with more operations is placed in the attribute compartment of another class. The attributes and parameters of the target class that access the determinant operation are also adjusted.
2. If two classes are in a one-to-many relationship, the determinant of the class on the “one” side is placed in the attribute compartment of the class on the “many” side. The attributes and parameters of the target class that access the determinant operation are also adjusted.
3. If two classes are in a many-to-many relationship, the determinants of the two classes are used as codeterminants to form a new class. The attributes and operations in the original classes and those dependent on those determinants are also moved to the new class.
4. If there is only one attribute for the determinant in a class, the class is deleted.

4. A Jigsaw Puzzle Concept for Image Search Model

Through the object-oriented technology and the use of the formalization of the object-oriented programming approach, the characteristics of the image (such as color, texture, and shape) were encoded into the database package as an object. The implementation of an image search model improved the artificial intelligence for the retrieval speed for a large amount of data [8,9].

The Search Method: The image search system provided various object images for multiple object queries, thus enabling the user to select one or multiple object images as the query image to retrieve the target image.

Setting up the Image Search System: (1) The rectangular salient images were derived through salient object detection. The SIFT algorithm was implemented to extract features from these images, and the BoF model was adapted to establish the BoF vectors of the images. (2) The image search system could use a combination of two to three object images as the query image to retrieve images that contain multiple objects. (3) The simulation system adopts the Euclidean distance to estimate the similarity between the object images and target images that were stored in a database. The target images and their corresponding names were then presented in descending order according to their similarity to the object images.

4.1 The Content-based Image Retrieval

The content-based image retrieval (CBIR) system is an image search solution based on visual feature extraction. It searches for images that are stored in a multimedia database that correspond to the contents of an image specified by the user. Figure 3 presents the image-matching process of a CBIR system. The CBIR system is described by common low-level visual features such as texture, color, and shape [32]. To improve the image retrieval precision, the bag-of-features (BoF) model based on the scale-invariant feature transform (SIFT) algorithm was integrated with salient object detection. These features were extracted from images to estimate the similarity between the query image and images stored in a database, through which the most similar images were retrieved. However, the features did not accurately describe the semantic concepts of images, as indicated by the summary of strengths and weaknesses of the features. Such features could be detected using SIFT, which was a computerized algorithm that could be implemented to detect and describe local features in images [33]. The SIFT algorithm has been experimentally verified to be the most sturdy local feature descriptor in the presence of geometric transformation in the field of visual identification, which was also one of commonly used methods of CBIR [34].

Figure 3. CBIR system for image-matching (photo courtesy of Khokher, A. and Talwar, R. [4]).
4.2 The Improved Experimental Environment

**Object Target Images:** This search system was adapted from Shiue, Y.-C., et al.'s experimental environment [3]. The improved image search system provides six types of object images: “house”, “car”, “ship”, “tree”, “bird”, and “flower”. The default object image was “house.” Each type of object image includes five relevant pictures. More object images could be incorporated into the system to retrieve different images containing different objects from the selected target images.

**Query Source Images:** Here, when an object picture listed on the object image panel was selected, the picture appears as the query image. A total of three object pictures could be selected as the query images. The image features and attributes in Figure 2 (object-oriented framework) and Figure 3 (CBIR system for image-matching) from the object-oriented technology that formats the object images are stored in the database and ready to use. After the query images were determined, the user clicks the “Search” button to begin searching for target images. Six types of image were extracted from the MSRA-A image data set [5], namely, “house”, “car”, “ship”, “tree”, “bird”, and “flower”. Each image type comprised 250 experimental images, which totaled 1,500 images for the experiment.

**Image Search Results:** After pressing the “Search” button, the object-oriented technology was calculated and combined with CBIR image-matching. By estimating the similarity between the query images and database images, the system listed the top 20 images according to similarity in the “Query Results” panel. If no target image was found, any picture from the search results that was deemed most similar to the target image could be selected as the query image to conduct further searches. Figure 4 presents the search results obtained using the “tree” and “house” type object images as the query image (such as a piece of jigsaw puzzle). A retrieved image could be chosen as the query image to improve the similarity of the search results according to the target image. Therefore, the object-oriented comparison program implemented the result of the multiple objects query.

5. Conclusions

Object normalization encompasses attribute and operation normalization. While attribute normalization can
be considered the normalization of the database in accordance with the actual class diagram to prevent discrepancies and duplication in the data, operation normalization still lacks a comprehensive discourse. First, on the basis of the previously mentioned principles for object-oriented normalization, the present study had proposed a comprehensive method and steps for object-oriented normalization that can be applied in the process of object modeling. This can maintain the object-oriented characteristics in the resultant class diagram and facilitate the consistency, completeness, and accuracy of data. Through the seven steps for object-oriented normalization, the concrete and control classes established in the class diagram allowed the system design to maintain its object-oriented characteristics. Second, the use of the formalization of the object-oriented programming approach used in the implementation of an image search model enhances the artificial intelligence for a large number of data’s retrieval speed. The graphical search system in this study was just an example. Therefore, the object-oriented system design can be expected to become even more popular and diversified in the future, and the incorporation of cross-disciplinary program languages or technologies is also foreseeable.

**References**


Manuscript Received: Nov. 27, 2017
Accepted: Apr. 16, 2018