Case-Based Reasoning for Design Patterns Searching System

Weenawadee Muangon
Software System Engineering Laboratory,
Department of Mathematics and Computer Science,
Faculty of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand
E-mail: s9062903@kmitl.ac.th

Sarun Intakosum
Software System Engineering Laboratory,
Department of Mathematics and Computer Science,
Faculty of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand
E-mail: kisarun@kmitl.ac.th

ABSTRACT
Design patterns are graceful solutions to specific software design problems. However, choosing proper design patterns for given software design problems might not be an easy task especially for novice developers. The design patterns searching tools are therefore needed to solve the problem. One major problem of the existing researches in this field is the indexing problem. This paper aims to solve the problem by presenting an elegant design pattern searching model that combines Case Based Reasoning (CBR) and Formal Concept Analysis (FCA) techniques. This model proposes a newly refinement technique. The technique allows experts to organize indexes to gain more complete software problem description in order to retrieve more appropriate design patterns. The indexes and cases similarity is calculated using FCA. The learning model to store new knowledge for retention process is also provided. Mean Average Precision (MAP) is used to assess the performance of the model. The preliminary experimental results show that the presented model has more retrieval ability in term of MAP comparing to the traditional model of CBR.

General Terms
Software Engineering, Reasoning, Knowledge Extraction, Knowledge Management.

Keywords
Design pattern, Design pattern retrieval, Knowledge Representation, Case Based Reasoning, Formal Concept Analysis.

1. INTRODUCTION
Design patterns are important techniques used to capture software design knowledge in order to solve software design problems. In general, design patterns are collected by experienced software designers [1] [2] [3] who formulate solutions in the specific context of recurring software design problems. Although design patterns are extremely useful in software design, choosing the right design pattern for a given design problem is a very difficult task. Especially, inexperienced software designers who have less knowledge of the field may not be able to cope with the large number of design patterns. For this reason, software design pattern searching tools for retrieving the right design patterns that can solve specific software design problems are considered helpful. Existing techniques for searching design patterns generally do not support efficient searching because of the major problem of indexing limitation. Usually, index assignment is performed by authors, which is difficult to match with the keywords of users. One way to solve this problem can be made by using a construction learning model. This paper proposes such a new model that applies both Case Based Reasoning (CBR) and Formal Concept Analysis (FCA). CBR is a smart knowledge learning model, which is a powerful tool used in problem-solving systems. FCA is a data analysis technique that can be used to discover hidden knowledge between indexes and cases in a case base. Utilizing the two approaches is very useful to solve the index limitation problem. This paper's model starts when a user inputs a new problem description. The system retrieves similar cases that are relevant to the problem by using a similarity function. Retrieved solutions are proposed to users and also stored to the system as new knowledge that can be reused for solving new problems. If users do not satisfy with the retrieved solutions, the system can find the right design pattern by giving alternative methods using FCA implementation. First, the structure of FCA enables discovery of related cases. The problem descriptions of related cases are presented to the user in order to extend comprehension of their problems. Second, FCA technique generates relevant indexes to make a more complete problem description. Lastly, using CBR retains new experiences in retrieval and revising processes as knowledge instrumental in solving similar problem in the future.

The remainder of this paper is organized as follows: Section (2) provides a concise survey of related work. Section (3) discusses important background knowledge to be applied to this paper's model. Section (4) presents a design pattern searching model. Section (5) proposes a simple prototype of the approach. Section (6) presents a preliminary evaluation of the model and explains the experimental results.

2. RELATED WORKS
This section briefly looks at some studies that relate to design pattern searching models. The goal of this section is to collect the studies in the field of design patterns retrieval. The studies in this field can be divided into two categories design pattern indexing and searching models.

2.1 Design Patterns Indexing
Research in [4] proposes formalizing technique to support design pattern retrieving system. It formalizes the text from intent section specified by [2] into phrases that are used to identify the roles of specific design patterns. The indexes that will be used in the searching model, discussed next, can be generated from these phrases. Research in [5] provided a way to find appropriate indexes to indicate the problem situation of design patterns. The research applied Pattern Component Markup Language (PCML) metadata of design patterns by adding pairs of subjects and predicates according to the software problem situation. This metadata can extend to a

design pattern search tool that uses verbs and subjects as queries. Both researches proposed a high performance method for design pattern indexing. However, a complete model to retrieve design patterns is not provided.

2.2 Design Patterns Searching Models

Researchers in [6] [7] proposed ReBuilder, a hybrid model of case based reasoning and WordNet ontology. The ReBuilder model represents a diagram of software application as a case and retrieves solutions by using a similarity metric of object-oriented elements. ReBuilder is a smart model which is easy to use and provides a clear solution. Unfortunately, while it focuses on retrieving related software applications, it ignores the problem-solving based on the context of software problem. Research in [8] offered a multi-agent system dependent on an implicit culture framework to search for solutions based on software design domains. This approach shares software design experiences between communities of users in order to choose the design pattern suitable for solving software design problem. This system allows previous knowledge to be used effectively; however, sharing unverified knowledge is the main weakness of this system. The design pattern search tool that is well-known for reliability is the Expert System for Suggesting Design Pattern (ESSDP) [9]. It uses a question-answering as knowledge given by human experts, and develops a user friendly interface to easily access the solution. The interface consists of a software design question and allows a choice to discover a more specific solution. This system is very reliable because the knowledge base is created by experts, but the knowledge of these human experts does not cover all the problems of that might be posted by users.

One way to solve the aforementioned problems is to use design pattern searching models that based on information retrieve (IR) [10] technique. Research in [11] focuses on index search and weight assignment. The vector space model is used to calculate the similarly calculates between queries and document. This model provides reasonable results, but the precision ratio is still considered low. Research in [12] adds a CBR approach o solve the problem. The research focuses on design pattern representation. It uses flat structure to represent a problem case of software design in which each case is described by the structure presented in [11]. The increase in precision percentage proves the efficiency of this model; however, using only flat structure to represent the case does not utilize the capability of CBR.

For this reason, the researches in [13], and [14], apply FCA method to discover embedded knowledge within a case base. However, the research is still lack of support for indexing problem. This research paper presents a more enhanced concept that can solve all previous mention problems as discussed in section 4.

3. BACKGROUND KNOWLEDGE

3.1 Design Patterns

Design patterns are reusable solutions that are used as an effective tool for solving the recurring software design problems. Experienced developers in the software design field found the common design problems that always occur in various design problems. Therefore, they collect and record the solutions to such problems in the form of reusable techniques and called them design patterns [1] [2] [3]. The design patterns provide efficient software design by not having to spend much time to find the solutions to problems that have already been solved. One of the books that provide collection of design patterns is GOF [2]. There are many studies that use GOF design patterns as a resource in their approaches. GOF collects 23 design pattern by dividing in to three categories, Creational, Structural and Behavioral. Creational aims to solve problems related to object creation. Structural focuses on creating a collection of related objects. Behavioral is a collection of design patterns that are used to capture behavior among related objects.

Generally, the GOF book describes a design pattern using the following template: *Pattern name*, *Category*, *Intent*, *Motivation*, *Applicability*, *Structure*, *Participants*, *Collaboration*, *Consequence*, *Implementation*, *Sample Code*, *Known uses* and *Related Patterns*. The proposed design pattern retrieval model focuses on sections that describe the problems and solutions of a design pattern. In sense the *Pattern name* and *Category* are the easiest sections that developers can understand a function of a design pattern quickly as possible. Further, research [4] [15] believed the *Intent* section is a shortest path to comprehend a design pattern. Hence, the training case base uses the information from these sections as cases problem descriptions.

3.2 Formal Concept Analysis Support in the Case Based Reasoning System

Case Based Reasoning (CBR) [16] [17] is a problem solving system that uses past experiences as knowledge used to solve similar problems. The knowledge is represented as a case that consists of problem descriptions and solutions. Several cases are collected into the case base. CBR processes consist of (i) retrieving the most similar cases from previous cases, (ii) reusing the retrieved solution to a new problem, (iii) revising the solution to adapt the result with precision (iv) retaining a new problem to use in future problem solving.

In order to carry out a CBR process, Formal Concept Analysis (FCA) [18] [19] [20] is applied to handle the knowledge in the case base. FCA is a mathematical approach that efficiently uses a data analysis method based on a concept lattice. FCA extracts dependent knowledge between attributes describing the objects. To implement FCA, a formal context is necessary. The formal context is defined as a triple *(G, M, I)*, where *G* is object set and *M* is attribute set, and *I* is incidence relationship $I \subseteq G \times M$.

From the formal context, FCA implements a set of concepts in which each concept consists of a maximal group of objects that are correlated with attributes. A formal concept formalizes the notions of extension and intension. The extension consists of all objects that share the given attributes, and the intension consists of all attributes shared by the given objects. Pairs of formal concepts might be ordered by the subset relation between their set of objects or the superset relation between their set of attributes. This is called the subconcept-superconcept relation, which is displayed as a hierarchy concept.

Belen et al. [21] proposed a preliminary method that combines CBR and FCA. They apply the ability of the FCA technique in the task of discovering knowledge embedded in the cases. In this research, each object is represented as a case and represents an attribute as an index. The notation specified in [21] is studied and the concept lattice of FCA is applied as an organization of the design pattern case base. From the FCA definition, the model can retrieve all cases that share indexes similar to the user's problem. Moreover, the system provides a more complete problem description by using the index dependency of FCA.

4. CBR FOR DESIGN PATTERN SEARCHING SYSTEMS

The CBR model in Figure 1 consists of two main sections. One is case base provision, which involves case representation, case indexing and case organization. Another is CBR engine, which includes four steps as follows.

1. Retrieve: A new problem from the user is given and a similarity function is used to retrieve a relevant case from the case base to solve the user's problem. Normally, a case in a problem-solving system consists of problem description and its solution. Hence, the retrieved results use a similar problem description and relevant solution.

2. Reuse: The retrieved solutions are proposed to the user and reused to solve the problem.

3. Revision: The precision of retrieved solution is improved by using a refinement technique to make a more complete problem description.

4. Retention: The new experience is stored as a new case in the case base after an analysis of the conditions of the learning process.

Before starting the model, case representation needs to be defined, which appropriates to represent experience in design pattern searching models.

4.1 Case Representation and Indexing

4.1.1 Case Representation

Case representation [22] is the most elementary issue in a case based reasoning model. For the problem-solving system, a case consists of a problem description and its solution.

1. Problem Description: In the CBR model, the relevant past cases are found by comparing the index of an input problem to cases in the case base. The information used to search the case base for matching cases is a concrete software design problem or a situation or problem that the user is experiencing.

2. Solution: In the CBR system, it is necessary to present a solution to a problem in order to reuse it. In this paper, a solution is a specific design pattern that should be applied to a problem.

CBR starts with a set of cases or case base training. To obtain a reliable case base, training cases are acquired from reasonable sources as discussed in section 6.2. Table 1 shows some cases in case base training of design pattern searching systems Table 2 shows examples of concrete software design problems and solutions.

In the next section, an approach is proposed for case indexing which index cases in order to be matched with similar cases in the retrieval process.

4.1.2 Case Indexing

Case indexing is a critical function that assigns indexes to cases for use in the retrieval process. In the previous section, a case representation of a problem-solving system was presented, which recorded in unstructured case or text forms. The main characteristic of retrieval in the CBR system is that it uses valuable indexes to identify successful cases which can be used to solve a user's problem.

In this section the methodology for converting text format into case structure for deployment of the indexing is presented.

Firstly, the stop word technique of the information retrieval model is used to formalize the indexing of unstructured cases. Stop words [23] are words that are proven to hinder indexer effectiveness (such as "is", "a", "an" and "the").

Secondly, an increase in the semantics of the case indexing is targeted. The feature extraction is applied to represent the characteristics of indexes that are used to identify problem descriptions.

Table 1. Design Pattern Description Cases

Problem Description (Design Pattern Descriptions)	***Solution***
Name *- Abstract Factory* ***Alias Name*** *- Kit* ***Category*** *- Creational* ***Intent*** *- Provide an interface for creating families of related or dependent objects without specifying their concrete classes.*	*Abstract factory*
Name - *Factory Method* ***Alias Name -*** *Virtual Constructor* ***Category*** *- Creational* ***Intent*** *- Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.*	*Factory Method*
Name *- Singleton* ***Category*** *- Creational* ***Intent*** *- Separate the construction of a complex object from its representation allowing the same construction process to create various representations.*	*Singleton*

Table 2. Concrete Software Design Problem Cases

Problem Description (Concrete Software Design Problems)	***Solution***
- Using tree structure to represent part-whole hierarchies	*Composite*
- Class notifies object to change state like public subscribe	*Observer*
- A wrapper that converts the interface of one component into another interface	*Adapter*
- The cursor that moves in different ways and represent the current position	*Iterater*

From careful analysis, seven typical features of software design problems are used for a case base. The seven features are (1) main name, (2) alias name, (3) category, (4) category function, (5) function, (6) entity and (7) keyword. Each feature is explained as follows:

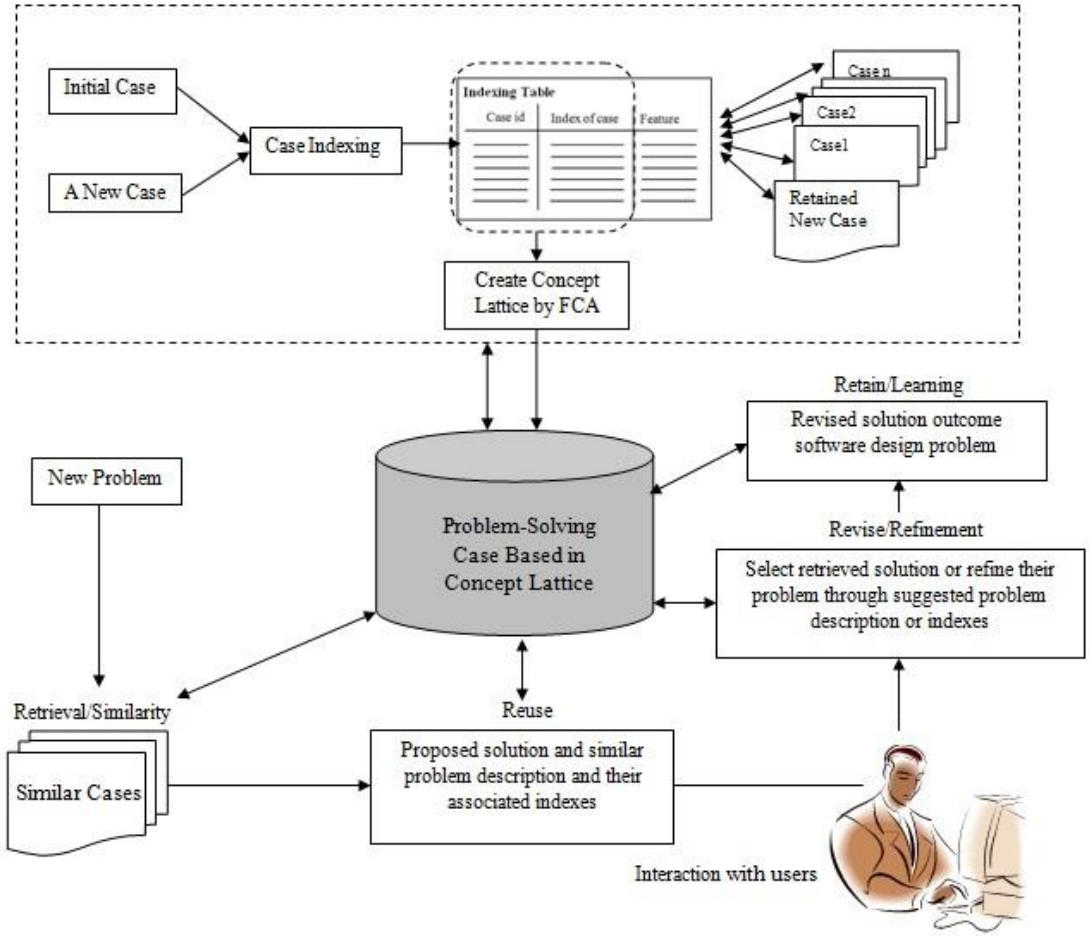

Fig 1: Architecture of CBR Model for Design Patterns Searching System

1. Main name (MA): represents an index that represents a specific design pattern name, such as abstract, factory, factory, method, adapter, or composite.

2. Alias name (AL): represents an index that refers to other names for the particular design pattern, such as kit, wrapper, public, or subscribe.

3. Category (CA): represents an index that represents a category of a design pattern, such as creational, structural, or behavioral.

4. Category function (CAF): represents an index that refers to major function of a design pattern category such as create, compose, or communicate.

5. Function (FUC): represents an index that refers to major functions or actions stated of a design pattern description, such as convert, represent, or build.

6. Entity (EN): represents an index that points to characteristics of object-oriented entities (object, class and interface), such as related object, singleton object, or abstract class.

7. Keyword (KW): represents an index that cannot be classified into previous features but has the ability to indicate a particular characteristic of a design pattern, such as different ways or various representations.

Table 3 shows an example of a feature extraction of a problem description. Currently, the constructing feature needs to be done manually since it requires the understanding of the semantics of each index. These features can possibly be used as a novel method to represent semantic information, in order to improve relevant case retrieval in the case based reasoning system.

Table 3. Features of Problem Description

	MA	AL	CA	CAF	FUC	EN	KW	SOLUTION
C1	Abstract Factory	kit	Creational	Create, Instantiate	Create	Relate Dependence families	Concrete	Abstract Factory
C2	Factory Method	Virtual Constructor	Creational	Create, Instantiate	Create	Interface	Instantiate	Factory Method
…								
C8	-	Wrapper	-	-	Convert Interface	-	-	Adapter
C9	-	Cursor	-	-	Move, Represent	-	Current Position	Iterator

Table 4. Formal Context

	Abstract	*Factory*	*Create*	*Family*	*Iterator*	*Compose*	*Wrapper*	*Structural*	*Relate*	*Instantiate*	*Virtual*	*Cursor*	*traverse*	*tree*	*subscribe*	*state*	…
C1	1	1	1	1	0	0	0	0	1	1	0	0	0	0	0	0	
C2	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	
C3	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	
C4	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	
C5	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	
C6	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	
C7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	
C8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	
C9	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	
…																	

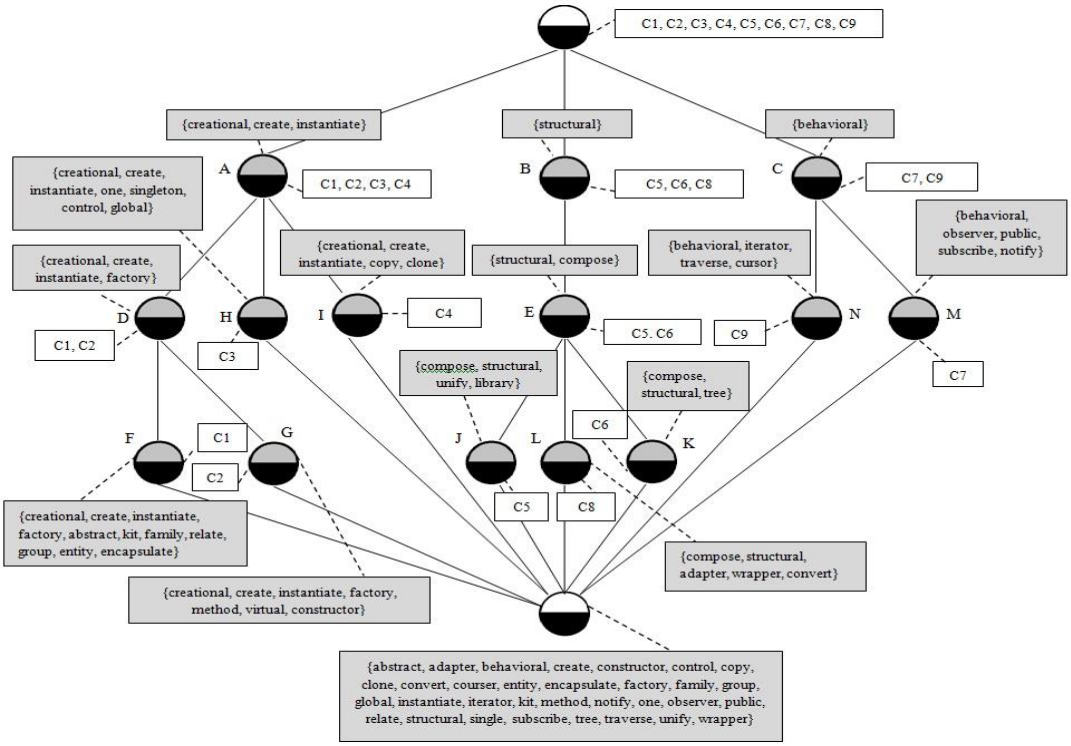

Fig 2 : An Example of Concept Lattice Case Base on Design Pattern Domains

4.2 Case Based Organization

In this paper, FCA is used to generate a concept lattice that elicits knowledge between cases and indexes from the case base. To demonstrate FCA implementation, a formal context of a problem description, described in the form of binary terms [0, 1], is used. Table 4 is an example of binary values between cases (objects) and indexes (attributes) of the design pattern case base. An index represented by "1" means it is indicative of a case, and "0" means it cannot point to any case. The binary values are transformed into a formal concept which is defined by a pair of a case set (extension) and an index set (intension). Extension of a concept means a case set which is shared by similar indexes and an intension is an index set that indicates the related cases. The final result of FCA implementation is a lattice structure that is generated from a formal concept. Figure 2 demonstrates an example of a concept lattice structure based on the design pattern case base. The FCA concept lattice helps to acquire knowledge indexing that is available in a case base. In this paper, knowledge indexing is suggested to refine the problem description for retrieval of a better solution. Moreover, the concept lattice relation overcomes certain index dependencies that prevent discovery of related cases. In the next section, the similarity technique, which corresponds to the concept lattice structure for retrieving and ranking the solution based on software design problem context, is presented.

4.3 Case Retrieval

In this section, a retrieval method that retrieves and ranks solutions to the user's problem is presented. According to the case base concept lattice in the previous section, the intension and extension properties of FCA are used as a critical argument in the similarity method. The intension of a concept represents an index set that indicates a problem description of a case. The extension represents a case set that is described by the intension. For example, the intension of concept *A* is {creational, instantiate, create} and the extension of this concept is {C1, C2, C3, C4}. In this paper, the retrieval process begins when a user enters a new problem into the system. A new problem is a problem description that the user requires a solution to. For finding the right solution, the index of a new problem matches with the case indexing of the case base. The system uses similarity function to rank the results. Equation (1) shows a similarity function that measures the similarity value between a new problem and a concept in a case base. Given two concepts *P* and *C*, *P* is a user problem concept, *C* is a case concept, and *I* is an intension which contains an index set of concepts in the case base. The value of each index in *P* and *C* is represented by "0" or "1". Following equation (1), the similarity assessment is a value between 0 and 1 where 0 is 0% similar, and the 1 means 100% similar.

$$Sim(P,C) = \frac{\sum_{i=1}^{n} I_{p,i} \cdot I_{c,i}}{\sqrt{\sum_{i=1}^{n} (I_{p,i})^2 \cdot \sum_{i=1}^{n} (I_{c,i})^2}} \qquad (1)$$

Where n is the number of indexes, $I_{p,i}$ is the i^{th} index of user problem concept and $I_{c,i}$ is the i^{th} index of concepts in the case base. The retrieved results are concepts which contain cases that related to the user's problem and the concept results are ranked according to the similarity value. In particular, a maximum value of similarity which shows at the first rank that could be obtained provides the relevant solution to the new problem. An example: given that a user problem is a free-text "interface create dependence object", the ranking of retrieved results are concepts {*A, F*} respectively by following its similarity values 0.8166, 0.2132. The real outcome of the retrieval process is the solution of cases in concept results which are presented to the user through a similarity ranking values. Observably, each retrieved concept involves several cases, but this does not mean there are several solutions

because many cases can hold a similar solution. In the next phase, the CBR model proposed an idea to adapt the precision of the retrieved results by using a revision process.

4.4 Case Revision

In this paper, the aim of the revising process is to increase the precision of the retrieval process. The retrieved results might be not satisfactory for the user because of the poor quality of an index of the user's problem description. To solve the aforementioned problem, this revising process offers related problem descriptions as an alternative way to narrow the scope of the user's problem and suggests an index that, when associated with the user's problem refinement technique, makes a more complete problem description. To create a successful revising process, two properties of FCA are applied as follows;

First, an FCA concept lattice classifies cases that shares indexes together. Using this property, the related cases that conform to indexes of the user's problem are clustered. In this paper, a concept node that is retrieved at the first rank (called the current concept) is used to lead to related cases. The related cases are found from extension of the current concept. Moreover, by definition, the extension of the superconcept contains a superset of cases in the current concept. Because all concepts of current concept and superconcept correspond with sharing indexes, hence, the system presents related cases that are achieved from extension of both concept types to the user. For example: Figure 2 concept *E* is proposed as {C5, C6}, the superconcept that is nearest to the current concept *E* is concept *B* which leads to {C8}.

Second, FCA method provides an index suggestion as a guideline to make a more complete problem description. The system starts from the current concept node that represents the best result of the retrieval process. In this section, subconcept nodes of the current concept are considered. Typically, a current concept contains matching and non-matching indexes. Although some indexes in the current concept do not match the user's problem, it still contains the closest words that are applied to refine the original problem of the user. Moreover, by definition, the intension of the subconcept is a superset of the current concept intension. Hence, the system suggests the intension of the current and subconcept to user. In this paper, a subconcept is used as the nearest to the current concept because it contains the maximum associated indexes that related to all cases of the current concept.

For example: Figure 2, a subconcept that is nearest to the current concept *A*, consists of three concepts, concept *D*, *H* and *I*. {creational, create, instantiate} is the intension of the current concept that points to concept *D*, *H* and *I*. Hence, in particular terms of each concept that should be suggested to the user, concept *D* suggests {factory}, concept *H* {one, singleton, control, global} and concept *I* {copy, clone}.

4.5 Case Retention

The aim of the retention process is to learn, through an experience, new knowledge instrumental into solving similar problems in the future. Normally, a new case comprises a new problem and its final solution. In this study, a new case from the retrieval and revising processes is retained as follows.

Retaining - Case Retrieval

First, for a proposed case retention from the retrieval process, a new case is generated when retrieval in the CBR operation is terminated. A new case represents concrete software design as a problem description and a retrieved solution that occurs at the first rank. This is the primary way that the user ends the operation after achieving the retrieved results.

Retaining - Case Revising

Second, the CBR system retains a refined problem description and its solution as a new case. The new case is created when the user refines the original problem by placing the index suggestion into the operation when finished.

Beside the two aforementioned approaches, the system needs design pattern experts to resolve a problem that cannot be solved by system operation alone.

5. AN ILLUSTRATIVE PROTOTYPE OF THE SEARCHING MODEL

The program prototype is developed for preliminary testing of the model. The program is as shown in figure 3. The search process is initiated when a user enters the description of a new software design problem. The results are retrieved and ranked based on the technique discussed in section 4. Moreover, the system returns related problems stored in the case base (as shown in the lower part of Figure 3) that are associated with the retrieved solution. These problems provide alternative ways to help users to gain more precise results.

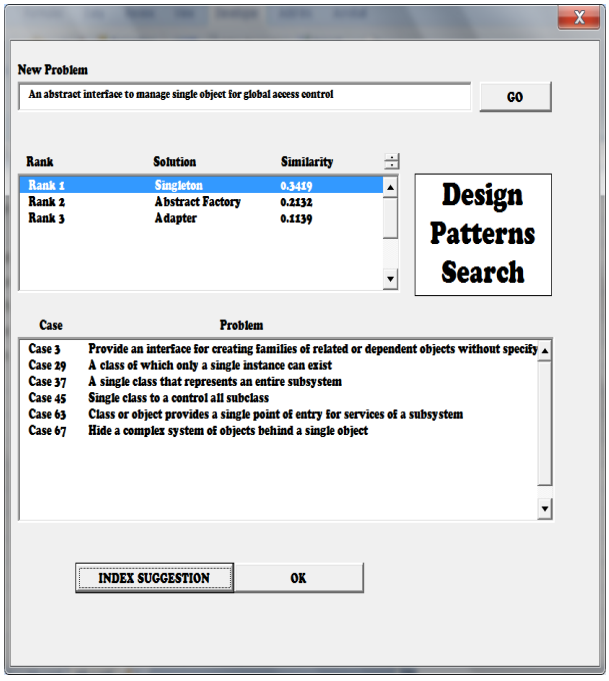

Fig 3: Design Pattern Retrieval Screen

Upon deciding whether the index should be used to refine the original problem, the system proposes a feature representative to assist in the index selection of the user. Note that the user in this step should be an expert who has strong knowledge in design patterns. The associated indexes are presented with their features. These features help users to select appropriate indexes to make a more complete problem description. In the implementation of the index suggestion, there are two forms to display as follows:

1. *Index-Feature*: This shows pairing between an index and its feature. This pairing assists a user to understand the importance of a suggested index. Figure 4 shows an Index-Feature suggestion screen that uses check boxes to receive and facilitate multiple selections by the user.

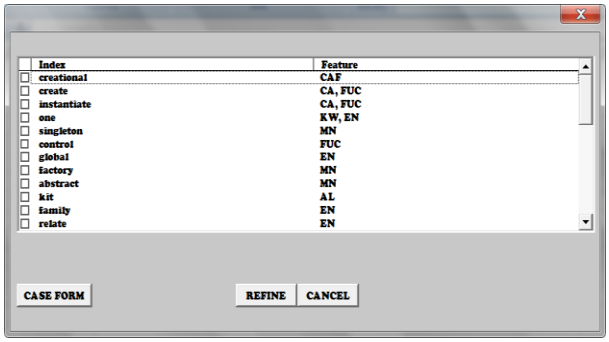

Fig 4: Index-Feature Suggestion Screen

2. Case Form: The Case Form presents indexes in form groups that correspond to related cases. This form makes it more convenient to choose indexes for problem refinement. Figure 5 shows a Case Form suggestion screen that uses a radio button to receive only one specified case.

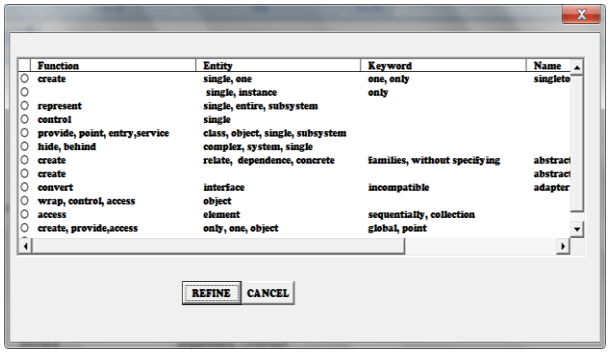

Fig 5: Case Form Suggestion Screen

6. EVALUATION AND EXPERIMENT

In order to prove the usability and efficiency of the approach used, a small experiment on the effectiveness of the model was performed.

6.1 Evaluation

The efficiency of results in the design pattern searching model are explained through mean average precision (MAP) calculation [10]. The average precision was computed by taking the average of the individual precision of the problem. MAP measures the quality of the retrieval system by averaging average precision from multiple problems. The mean average precision is defined as shown in equation (2).

$$MAP = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{Q_j}\sum_{i=1}^{Q_j}\Pr ecision(C_{i,j}) \qquad (2)$$

where N is the number of problems, Q_j is the number of relevant cases for problem j and $Precision(C_{i,j})$ is the precision value at ith relevant case.

6.2 Training and Test Cases

For testing, two data case sets are provided. First, available training cases are used for preliminary knowledge. Second, a set of test cases to test retrieval performance of the model are built.

Training Case Set involves 72 cases achieved from two data sources. First, design pattern description from the most famous design pattern book GOF [2]. There are 23 design patterns described in GOF and they are utilized as preliminary data in the model. The remaining 49 cases come from concrete software design problems that were solved by design pattern techniques discussed in GOF. They are used as problem descriptions for data training purpose. In summary, entire training set of cases were provided by design pattern experts that analyzed the quality of the problem description and its solution before retention in the training case base.

Test Case Set is a data set achieved from 21 design pattern beginners who volunteer to test the model. The test case is divided into two sets. Test case A involves 30 cases and Test case B consists of 33 cases. Both Test case sets were used to test the retrieval, performance, revision and retention processes.

6.3 Experiment and Result Discussion

The following two experiments demonstrate the retrieval performance of the design pattern searching model.

Experiment 1 – Problem Refinement

This experiment evaluates the retrieval performance in the revision process that enables comparison of the precision between the results of the original problem and the refined problem which is adapted by the index suggestion. The testing process used 30 cases in Test Case A to view the advantage of suggested indexes. Table 5 presents the retrieval performance before and after the revision process in the model. The MAP percentage increases in the refined problem means a significant advantage in the index suggestion.

Table 5. Problem Refinement Performance

No. of Cases in Test Case A	MAP (%)	
	Original Problem	Refined Problem
30	58.76%	77.45%

Experiment 2 – Case Base Learning

In the CBR model, the retention process provides the ability to learn from useful cases and use this information to solve new problems. In this section, Test Case A is used as the useful cases that are added into the case base. Test Case B is used to test the performance of the retrieval process before and after the Test Case A is added. The results in table 6 shows that the MAP percentage of the retrieved results for the case base after adding the test case A (Case Base Learning) is better than the original case base (Case Base Training).

Table 6. Case Base Learning Performance

	No. of Cases in Case Base	Test Case B MAP (%)
Case Base Training	72	63.57%
Case Base Learning	102	76.34%

7. CONCLUSIONS

In this paper a novel design pattern searching model is presented. The proposed model integrates Case Based Reasoning and Formal Concept Analysis to solve the indexing problem persists in the existing researches. The refinement technique is developed to help organizing the indexes to gain more complete problem description that leads to more precise results. The model also provides a learning method to retain new experiences in order to solve similar problems that might be entered into the system in the future. A simple program prototype is developed to evaluate the model. The efficiency of the model is measured by mean average precision. The results are shown that the performance of the proposed model is better than the normal case base reasoning model.

8. REFERENCES

[1] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects. John Wiley & Sons, 2000, vol. 2.

[2] E. Gamma., H. Richard., R. Johnson and J. Vlissides, Design Pattern: Elements of Reusable Object Oriented Software, Addison-Wesley, 1995.

[3] F. Buschmann, K. Henney, and D. C. Schmidt, Pattern-Oriented Software Architecture: On Patterns and Pattern Languages. John Wiley & Sons, 2007, vol. 5.

[4] G.Andreas,andE. Mattias. "Formalizing the Intent of Design Patterns", An Approach Towards a Solution to the Indexing Problem Technical report 1999-006, Uppsala University. 1999.

[5] P. Daniel, and C. Gyorgy. "Design Pattern Matching", Periodica Polytechnica Ser. el. Eng. Vol. 47. No. 3-4. 2003. pp. 205-212.

[6] P. Gomes, F.C. Pereira, P. Paiva, N. Seco, P. Carreiro, J.L. Ferreira, C. Bento., "Combining case-based reasoning and analogical reasoning in software design", In: Proceedings of the13th Irish Conference on Artificial Intelligence and Cognitive Science (AICS'02), Limerick, Ireland, Springer-Verlag 2002.

[7] P. Gomes, F.C. Pereira, P. Paiva, N. Seco, P. Carreiro, J.L. Ferreira, C. Bento., "Case retrieval of software designs using wordnet", In Harmelen, F.v., ed.: European Conference on Artificial Intelligence (ECAI'02), Lyon, France, IOS Press, Amsterdam, 2002.

[8] B. Aliaksandr., B. Enrico., G. Paolo. "Choosing the right design pattern: the implicit culture approach", In: Proc. of the Workshop on Multi-Agent Systems and Simulation at the Industrial Simulation Conference 2006 (ISC-2006), 2006.

[9] M. Gary, S. Abhijit, F. Daniel J. "An expert system for the selection of software design patterns", Expert system Journal Volume 23, Issue 1, February 2006, pp. 39-52.

[10] B. Yates R., and R. Berthier., Modern Information Retrieval, Addison Wesley, 1999.

[11] W. Muangon. and S. Intakosum. "Retrieving model for design patterns", ECTI Transactions on computer and information technology, Vol. 3, No. 1, May 2007, pp.51-55.

[12] W. Muangon. and S. Intakosum. "Case-Based Support Retrieval for Design Patterns", JCSSE International joint conference on computer science and software engineering, Vol. 1, May 2008, pp. 419-423.

[13] W. Muangon. and S. Intakosum. "Retrieving Design Patterns by Case-Based Reasoning and Formal Concept Analysis", ICCSIT International Conference on Computer Science and Information Technology, Vol. 4, August 2009, pp. 424-428.

[14] W. Muangon. and S. Intakosum. "Adaptation of Design Pattern Retrieval Using CBR and FCA", ICCIT International Conference on Computer Science and Convergence Information Technology, November 2009, pp. 1196-1200.

[15] H. Kampffmeyer and S. Zschaler, "Finding the pattern you need: The design pattern intent ontology", in MoDELS, ser.Lecture Notes in Computer Science, G. Engels et al., Eds.,vol. 4735. Springer, 2007, pp. 211-225.

[16] A. Aamodt. and E. Plaza. "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches", Published in: AI Communications, Vol. 7 Nr. 1, March 1994, pp 39-59.

[17] R.L. De Mantaras et al., "Retrieval, reuse, revision, and retention in case based reasoning", Knowledge Engineering Review, vol. 20, pp. 215-240, 2005.

[18] R. Wille, "Formal concept analysis as mathematical theory of concepts and concept hierarchies", Formal Concept Analysis: Foundations and Applications, LNAI 3626, Berlin: Springer, 2005, pp. 1-33.

[19] S.O. Kuznetsov, S.A. Obiedkov, "Algorithm for the construction of concept lattices and their diagram graphs", In Proc. of the 5th Principles of Data Mining and Knowledge Discovery: European Conference, Freiburg, Germany, September 3-5, LNCS 2168, Berlin: Springer, 2001, pp. 289-300.

[20] U. Priss, "Formal concept analysis in information science", Annual Review of Information Science and Technology, vol. 40, pp. 521-543, 2006.

[21] D. Belen, M.G. Antonio, P.G. Pablo and A. Pedro, "Formal concept analysis for knowledge refinement in case base reasoning", In Proc. of the 25th International Conference on Innovative Techniques and Applications of Artificial Intelligence, 2005, pp. 233-245.

[22] B. Ralph, J. Kolodner and E. Plaza, "Representation in case-based reasoning", Knowledge Engineering Review, vol. 20, pp. 209-213, 2005.

[23] T. Rachel, H. Ben, O. Iadh. "Automatically building a stopword list for an information retrieval system", J Digital Informat Manage, 2005, pp. 3-8.