Construction of the Dependence Matrix Based on the TRIZ Contradiction Matrix in OOD

Jianhong Ma, Quan Zhang, Yanling Wang, and Tao Luo

School of Computer Science and Engineering, Hebei University of Technology, Tianjin, 300130, China
{Jianhong.Ma,Quan.Zhang,Yanling.Wang,Tao.Luo, majianhong}@scse.hebut.edu.cn

Abstract. In the Object-Oriented software design (OOD), design of the class and object, definition of the classes' interface and inheritance levels and determination of dependent relations have a serious impact on the reusability and flexibility of the system. According to the concrete problems of design, how to select the right solution from the hundreds of the design schemas which has become the focus of attention of designers. After analyzing lots of software design schemas in practice and Object-Oriented design patterns, this paper constructs the dependence matrix of Object-Oriented software design filed, referring to contradiction matrix of TRIZ (Theory of Inventive Problem Solving) proposed by the former Soviet Union innovation master Altshuller. As the practice indicates, it provides a intuitive, common and standardized method for designers to choose the right design schema. Make research and communication more effectively, and also improve the software development efficiency and software quality.

Keywords: TRIZ, contradiction matrix, Object-Oriented Design, dependence matrix.

1 Introduction

The principles that software design emphasizes are well supported by the OOD, but during the process of developing, designer often are confused, or can not easily decide which design schema to be chosen. For instance, the most difficult point is how to find the objects and how to ascertain the dependent relationship between them in OOD. Too much dependence is disadvantage for modular design and reuse of class. The class is more independent, the reusability is better. That more dependent relations lead to the sensitivity of design change is more obvious which made modification, maintenance and test more difficult. So in practice, we need to find a compromise design schema with a lot of mending of some dependent relations.

Get the intention behind the design by allocating the collaboration and responsibilities between the objects with design patterns. With pattern design, designer can improve and reuse the existed successful designs and structures easily, and make it clear

R. Tan, G. Cao, and N. León (Eds.): CAI 2009, IFIP AICT 304, pp. 219–230, 2009.

to the following system designer. Design pattern can increase the flexibility and the reliability of a system, and it can also make the communication easier inside the group, or between groups. With the increasing patterns, how to choose a best one for the system has become a bottleneck, especially for the designer unfamiliar with it. So it is essential to choose an easy and effective pattern.

It is necessary to provide a common, standardized method to describe the problems that occur in the process of building classes, objects and provide corresponding improving schema. It will make research and communication between designers easily, and enhance the development efficiency and performance. This paper constructs the dependence matrix, referring to the theory of contradiction matrix of TRIZ.

2 TRIZ and the Application in Software

2.1 Introduction of TRIZ

TRIZ meaning "The theory of inventive problem solving", was developed by a Soviet engineer and researcher Genrich Altshuller and his colleagues starting in 1946. After analyzing about 2.5 million high-level patents, Genrich Altshuller and his group concluded the laws of technical system evolution and the principles of how to solve technical contradiction and physical contradiction. Uniting with the laws and principles of other fields, they built a comprehensive theoretical system with methods and algorithms of solving problem and completing innovation. The theory was kept a secret to other countries, before the middle of 1980s. But it was introduced gradually to the production design filed with the emigration of scientists to the western countries afterwards, producing enormous effectiveness.

In TRIZ relative research document, contradiction matrix is one of most useful concepts and tools. It includes 39 feature parameters and 40 inventive principles. The 39 feature parameters usually occur in technical contradictions, some of which depend on whether the object is static or dynamic. To find the best principle, The 39 feature parameters are arranged into the rows and the columns (Rows of the matrix indicate the 39 feature parameters that typically to be improved, and the columns are affected parameters). Each matrix cell points to principles that have been most frequently used in patents in order to resolve the contradiction. Analyzing the most important inventions of the global patent database, they got the principles. The train of thought of which using TRIZ contradiction matrix to solve the problem is: The question of innovation turns into a routine problem, and then rapidly access to innovative solution with the experience of the previous. Fig.1 is a schematic drawing which shows the steps of resolving the TRIZ problems.

For Instance, Make a stronger system by increasing the length of the moving object in the mechanical problem, but the weight is more. As the contradiction matrix indicates, the principle to solve the problem of the increasing weight because of costing more material is to compensate weight, make it dynamic, liquidize, abandon, mend and use hydraulic pressure structure. And then according to the practical problem to select the most suitable scheme to solve the problem.

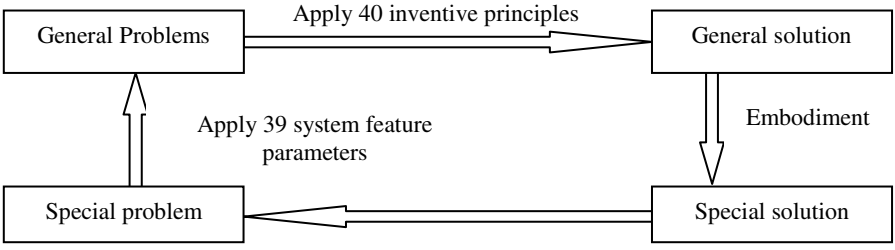

Fig. 1. Schematic Drawing of Resolving the Innovation Problem

2.2 The Research and Development of TRIZ in Software Field

The theory of TRIZ is used in engineering generally. At present, it is introduced to other filed such as natural science, social sciences, management science, bio-science and so on. The cases of using the 40 principles to solve the problems in Industry, architecture, micro-electronics, chemistry, biology, sociology, medicine, food, business, and education have been found. And the cases were used to guide the solution to problems in various fields recently.

TRIZ has been introduced into the information technology recently, and has solved some problems that attract people's attention.13 patents have been applied relative with it. Kevin C.Rea indicated the relation between the 40 principles and the computing filed [1][2] and solved the problems of concurrent programming, the access and data distribution of the shared memory[3]. Ellen Domb and John W.Stamey proposed the contradiction matrix of TRIZ could be used to solve the contradiction problems in the structure system (time and space) and finds strong links between the 7 design patterns of GOF and a specific set of TRIZ principles [4]. Garikapati Pavan Kumar proposes an innovative application of TRIZ and Six Sigma for software process improvement. And it specifies how to use contradiction matrix and 40 principles to improve the solutions for the problems [5].

Based on the previous work, this paper proposes a common, standardized method to describe the dependent relations between the classes and objects in the process of software design, and much more schemas are studied and proposed to solve designers' design problems. This paper is organized as following:

1. Extraction of feature parameters in OOD, based on researching class and object's design and the dependent relationship of the classes and the objects, the feature parameters of the dependence matrix are extracted, referring to the process that the 39 feature parameters are extracted in the TRIZ contradiction matrix.
2. Construction of the dependence matrix in OOD, Ascertain the relationship between design patterns and their corresponding solutions by analyzing the each pattern and each problem in the OOD. Then referring to the principle of that the contradiction matrix be constructed in TRIZ, construct the dependence matrix in OOD.
3. Develop the computer-aid software design tool, classify and store the patterns into the database. The tool supports the dynamic growing and evolution, easily retrieved to find the required design schema for designer.
4. Verification of the constructed dependence matrix with practical example.

3 Construction of Dependence Matrix in OOD

In the process of the OOD, the most difficult point is how to find the objects and ascertain the dependent relationship between them. Too much dependence is disadvantage for modular design and software reuse. The more dependent relations are, which lead to the sensitivity of software requirement changes, the more difficult the software is modified and tested. In practice, when we chose a schema to improve one performance, the other performances or requirements may be not satisfied. So, we need to find a compromise design schema to adjust some dependent relations.

Design pattern can help the designer to make a new design based on previous work and to reuse the previous successful design schema. Using design pattern to solve specific problem makes the OOD more flexible, elegant and also makes the reusability better. In this paper, we treat the design patterns as the primary design schemas.

The dependence of dependence matrix not only refers to coupling relations of object, class and interface but also other relations of parameters. Such as affected and restricted design parameters in the process of improving them. The rows of the dependence matrix stand for the improving parameters, and the columns stand for the affected or restricted design parameters in the process of improving the design parameters and the parameters having dependent relations with the improving parameters.

Construction of dependence matrix is divided into the following two-step: First, make certain the bound of research, and conclude and classify the problem happening in the process of designing classes and objects in OOD, and extract the feature parameters that affect the software performance. Then, we construct the dependence matrix in OOD, and treat the design patterns as the design schema.

3.1 Feature Parameters Extraction in Software Design

By analyzing large quantity of process of OOD, we can find out the advantage and disadvantage from the existed design schemas and the concrete problems in OOD. We also can get feature parameters through the abstract and summary of these problems.

1. Getting feature parameters from object-oriented coupling relations.

In the system of OO, coupling means the relationship between classes. Coupling is a two-edged sword: on the one hand, the interactions, which we called coupling which between classes come into being the functions of one system, on the other hand, due to the existence of coupling, the change of an object or module could lead to the change of other objects or modules. Coupling is too strong to reduce the quality of the whole program sharply, so it is necessary that we recognize objects coupling accurately in system and decouple appropriately. Coupling includes generalization coupling, realization coupling, correlative coupling and dependent coupling in OOD.

Generalization coupling: It means the subclass or interface inherits properties or operation of its parents. When a class generalizes (inherits) many types at the same time, it is possible that: Its mutual effects between the properties and operations will gets bigger, which derived from the parent class. It is usually that multiple generalizations (inheritance) should be avoided as far as possible. So the design parameters that are inductive from generalization coupling include: Subclass, parent class, interface,

properties, functions and the levels of classes. Here, the level of classes means the levels of some class that derived from relation trees. And it means, if the more levels of class have, the more classes inherit the members from the parents, and it also means it is more difficult to forecast the behavior and more complex in the design and maintenance, then it is good in reusable respect.

Realization coupling: It means one object has too many interfaces and classes to realize. The object needs to be decomposed for the mass tasks. So we can conclude the feature parameter: the number of the child class.

In the same way, we can obtain the corresponding feature parameter from the correlative coupling and dependent coupling.

2. Getting feature parameters from existed design patterns.

For instance: If we adopt the Simple Factory Pattern, it can make the customer get the instances of classes according to parameters, so it not only avoids instantiate class directly, but also made coupling more loosely. However these class, which could be instantiated, are determine in compiling duration, if we add new class, it will need to modify the factory code as well. Above all, we can find out the advantage and disadvantage of Simple Factory Pattern from this design pattrens: The advantage is: To avoid instantiate class directly and made coupling more loosely. The disadvantage is: Add new class will lead to modify the factory code. If we extract these furthermore, we can obtain feature parameters in OOD: Create class, Dependency relationship between classes and Add class.

3. Getting feature parameters from concrete problems in OOD

For instance, Suppose we need analogue the situations of one person wearing hat in one software system, which hat type should be choose depend on the condition, the one's like and other factors. The aim of the design is to confirm the type of the hat when one person present in a specific condition. So, according to the requirement, it forms a simple model as Fig.2 shows.

As it shown in Fig.2, If we want to add "climbing" based on the existed behaviors "riding" and "swimming", the common practice is to build a sub-class to extent behavior, but if we add a new behavior "running", We will find an apparent drawback in this design schema: One class extent a function inherited from the other class, but actually there is no logically inherited relationship between them. Meanwhile, it also increases the levels of the classes. So, we can conclude the feature parameters from this concrete problem: the combination of classes and the level of class.

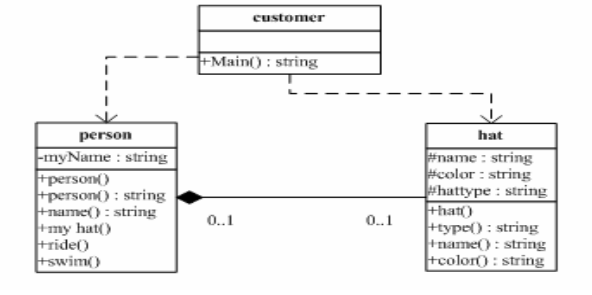

Fig. 2. Model of Class

Beginning with the classes, objects, interfaces, methods and their dependent relationship, this paper abstracts the feature parameters in OOD. The parameters are depicted in Fig.3.

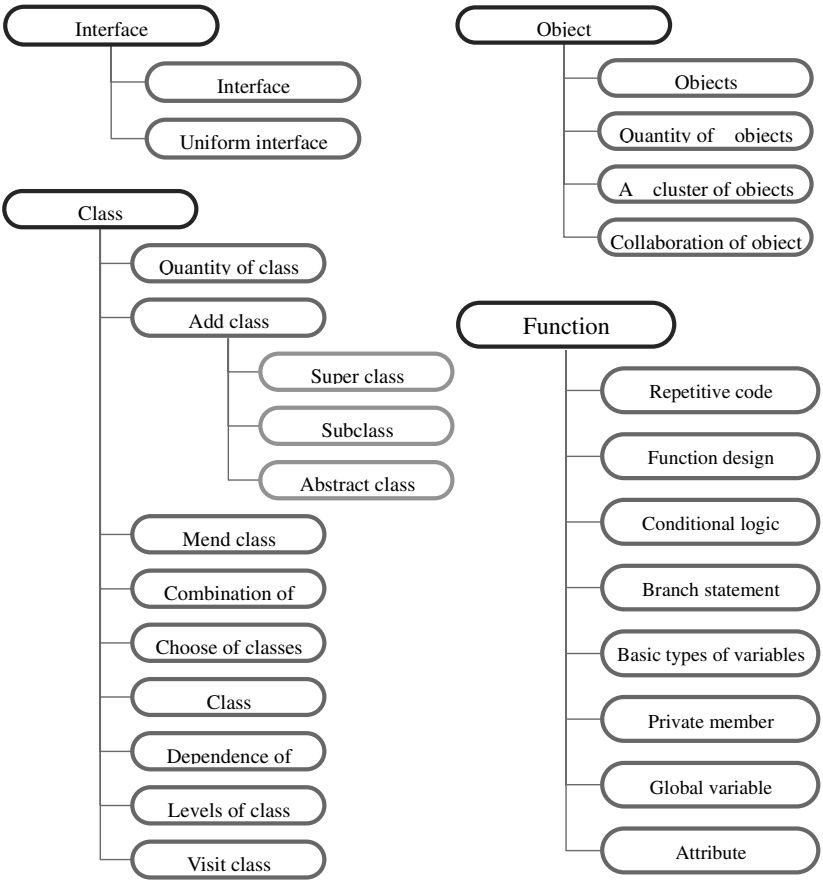

Fig. 3. Feature Parameters

3.2 Construction of Dependence Matrix

According to the principles of the construction of TRIZ contradiction matrix, we construct dependence matrix of the field of software. We create the 24X24 dependence matrix by using the 24 feature parameters are summarized above. The feature parameters will be improved as columns and the feature parameters will be affected or dependent parameters as rows and writes them into the dependence matrix. Rows and columns of cross-points constitute a pair of interdependence's relation. On the process of the specific software design, First of all, using these features to describe the

problem and to find the interdependence of the parameters. Then find the design schemas in the corresponding grid of the matrix. Choose mature design patterns, design idea and design prototype as design schemas of the cross square grids, For instance, the 23 species of design patterns, component technology, collaboration, encapsulation and so on. We can write their corresponding codes into the dependence matrix based on deeply understanding the solution. The design schemas in this paper mainly are based on the 23 species design patterns are put forward by GOF. Specific examples are as follows:

Object structural pattern—composite pattern.

Intent: Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Motivation: The user can group components to form larger components, which in turn can be grouped to form still larger components.

Applicability: It can represent part-whole hierarchies of objects; Clients can ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.

Consequences: Primitive objects can be composed into more complex objects, which in turn can also be composed; It can make the clients simple. Clients can treat composite structures and individual objects uniformly; Makes it easier to add new kinds of components.

By means of analyzing intent, motivation, applicability and consequences of combination pattern we can summarize and abstract these: 1. Combine objects that have uniform interfaces to form a larger object to achieve more functions. 2. Add new classes easily on the foundation of adding a new generated subclass. So write the corresponding code 8 of combine pattern separately into the square grids of object composition-uniform interface and add class-quantity of classes.

Table 1. Dependence matrix

		Affected or dependent parameters			
		Add class	Object	Combination of classes	Function
Improving parameters	Add class			8, 9, 18	
	Object	1, 3, 4	1, 3	4	12, 14, 20, 21, 23
	Class	1, 5, 3		6, 15	

According to analyzing and understanding the process of which how design patterns fill the dependence matrix, improve dependence matrix based on understanding each design pattern. At last we can construct the dependence matrix that can guide and optimize OOD. The partial structural diagram of the dependence matrix is shown in Table 1.

In the table, the content of second column indicates the parameters will be improved and the second row indicates the parameters that are affected or need to be depended in the process of improving. The numbers in the table indicate which design schemas can solve the problem. Partial contents of this design schema could be retrieved in Table 2. The design schemas have the equal location and they could be free to choose by people who use it to solve the practical problem.

For instance: Algorithm (function) is expanded, optimized or replaced in the process of development and reuse frequently. The object which depends on a special algorithm (function) has to change when the algorithm change. In the dependence matrix, we choose the "function" in column and choose the "object" in row. The design schemas are: 12, 14, 20, 21, 23. The concrete design schemas are bridge pattern, iterator pattern, strategy pattern and template method pattern. Finally we choose a best design schema referring to the practical problems.

Table 2. Design Schemas

No	Design schema	Explanation
1	Factory method	create a single entity
2	Singleton pattern	create a secure object exactly
3	Abstract factory pattern	Create a cluster of objects
4	Prototype pattern	Involving "mixed and matched"
5	Builder pattern	Construct complex object
6	Façade pattern	Provide interface for the collection of objects
7	Decorator pattern	Increase object in the run-time
8	Composite pattern	Express tree structure of the object
9	Adapter pattern	Simplify the use of external feature parameters
10	Flyweight pattern	By using multiple examples to minimize space consumption
11	Proxy pattern	Provide a surrogate or placeholder for another object to control access to it.
12	Bridge pattern	Abstract and realization of separation
14	Iterator pattern	Visit the set of elements
20	Strategy pattern	Make algorithm implementation independent on its customers
21	Template Method pattern	Make subclass which don't change the structure of algorithm can re-define some specific steps of the algorithm.
22	Memento pattern	Capture the internal state of an object and save the state out of the object.
23	Visitor pattern	Through a unified interface to visit different types of elements of the operation

4 Application of the Dependence Matrix

The concrete process of using dependence matrix to solve the problems in the process of OOD is as Fig.4:

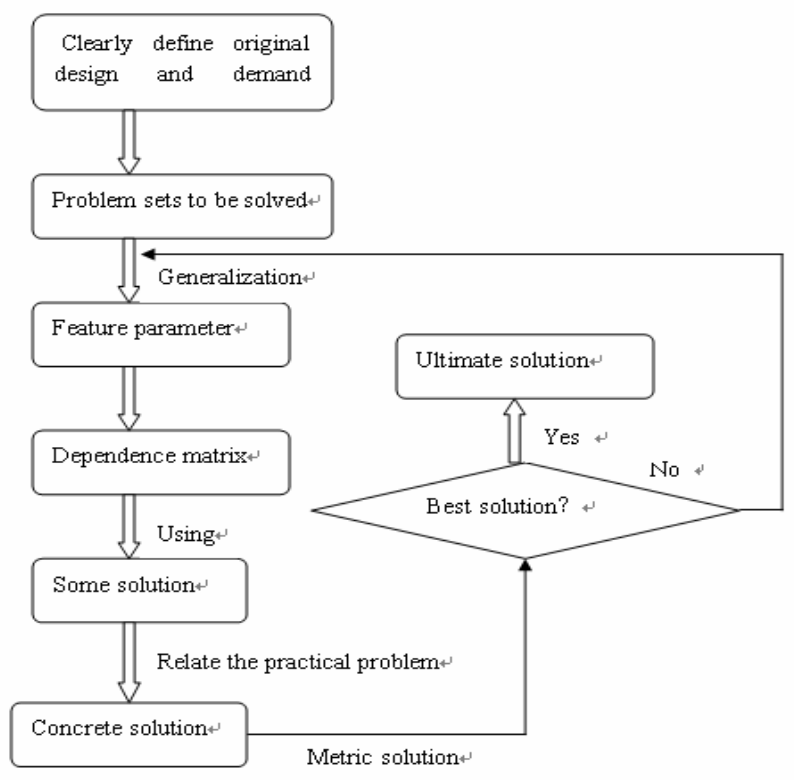

Fig. 4. Flow Chart of Solving Problems

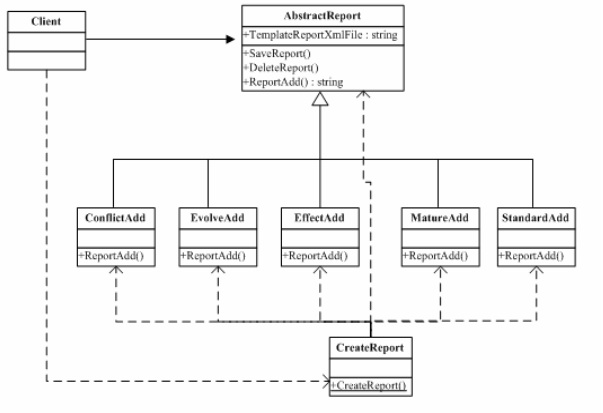

Fig. 5. System Structural Diagram

The following is to show the concrete application of the dependence matrix by depenenceMatrix software developed by C# as an example. The innovative design system mainly contains five modules: conflict, evolve, effect, mature, standard. All of the modules include the performance function: report generation. Each concrete

methods of report generation are different. In this system we define an interface to create an object which can visits each report generation module. The System structural diagram as Fig.5 shows.

After completing the procedure, the new question appears. Object need to be created before visited. Yet who can create the object? If the customer creates the object, the concrete implementation constraints the specified class name when create the object. Finally, the dependent coupling between objects is produced, which makes the future change more complex. Hence we need improve the design schema to realize decoupling. We can use the dependence matrix that is created above to solve the problem. Specific interface as Fig. 6:

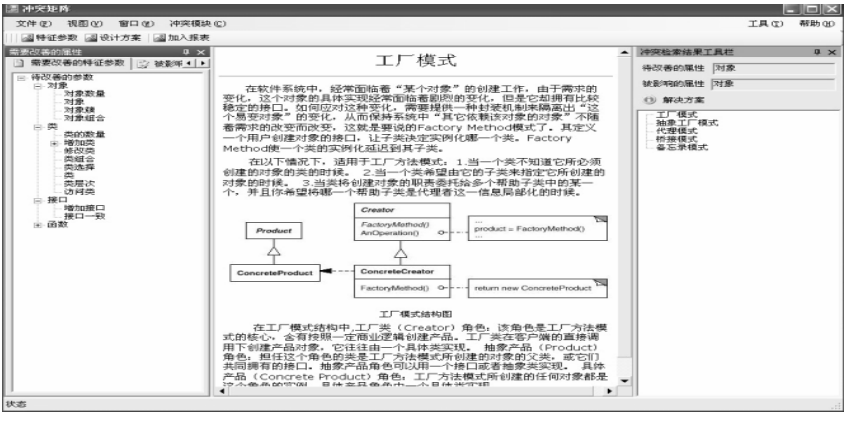

Fig. 6. System Interface

First of all, choose the "object" in the "improving parameters". Select the" object" in the "affected and dependent feature parameters". So we can gain the corresponding design schemas: 1, 3, 11, 12, 22, and they are factory method, abstract factory pattern, proxy pattern, bridge pattern and memento pattern. According to the concrete situation, we choose the factory method. Specific solutions are as following:

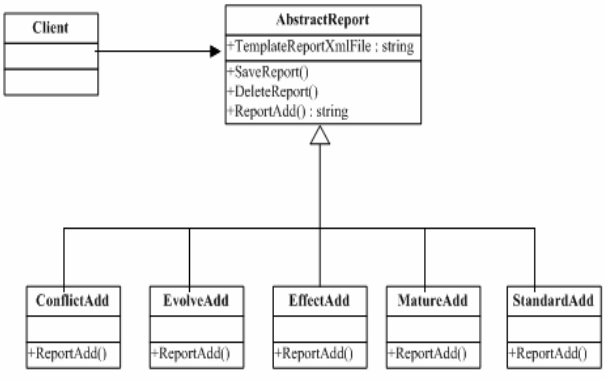

Fig. 7. Structural Diagram

We add a class and use it to create an object collaterally, and to do like this, customers do not need to know about that this object belongs to which specific subclass. In other words, the dependence on the concrete implementation is replaced by the dependence on the concrete interface, which improves the design schema. The specific structure as Fig 5 shows.

5 Conclusion

After a large number of researches and data collations, we have constructed the dependence matrix with the feature parameters extracted from the summarized and concluded problems in the OOD .And according to the constructed theory of 40 invention principles and contradiction matrix of TRIZ, treating the design patterns as the primary design schemas of the dependence matrix, we illustrated how to use the matrix. We will do our research in the following two directions in the future: first, continue to research on the feature parameters of the matrix, make them scientific and common. Second, research and improve design schemas that can solve the problems. Design schemas do not limit to the 23 design patterns. We can combine or extent the design patterns, and we can add the aspect-oriented or service-oriented as design schemas into the matrix.

Acknowledgments

This research is supported in part by the Key Project of the Ministry of Science and Technology of the People's Republic of China under Grant Numbers 2008IM030100, and the science and technology key project of Hebei Province under Grant Numbers 09212102D. Any opinions or findings of this work are the responsibility of the authors, and do not necessarily reflect the views of the sponsors or collaborators.

References

1. Kevin, C.R.: TRIZ and Software 40 Principles Analogies, Part 1. The TRIZ Journal (September 2001), http://www.triz-journal.com/archives/2001/09/e/index.htm
2. Kevin, C.R.: TRIZ and Software 40 Principles Analogies, Part 2. The TRIZ Journal (November 2001), http://www.triz-journal.com/archives/2001/11/e/
3. Kevin, C.R.: Using TRIZ in Computer Science Concurrency. The TRIZ Journal (August 1999), http://www.triz-journal.com/archives/1999/08/d/
4. Domb, E., John, W.S.: Describing Design Patterns in SoftWare Engineering. The TRIZ Journal (February 2007), http://www.triz-journal.com/archives/2007/02/02/
5. Kumar, G.P.: Software Process Improvement –TRIZ and Six Sigma. The TRIZ Journal (April 2005), http://www.triz-journal.com/archives/2005/04/07.pdf
6. Fulbright, R.: TRIZ and Software Fini, http://www.triz-journal.com/archives/2004/08/02.pdf

7. Dana, G.M.: 40 Inventive Principles with Applications in Education, http://www.triz-journal.com/archives/2004/04/04.pdf
8. Zhong, J., Gu, L., Meng, H.: A Method of Automatic Selection for Software Design Patterns. Computer Technology and Development (9), 21–27 (2007)
9. Bai, D.Q., Chen, Q.: Application of the TRIZ Contradiction Matrix in Service product innovation process. Innovation and Knowledge Management Symposium Papers, 533–544
10. Lei, D.: The research of TRIZ innovative theory introduced to management of software program. Science and Technology Management Research (2), 107–109 (2006)
11. Gao, C., Huang, K., Zhang, Y.: The Contrast and Application of the Problem Resolving Tools of TRIZ. Machine Design and Research (22), 13–15 (2006)
12. Ji, C.: Study of De sign Pattern of Software and it s Application. Journal of Shanghai Dian Ji University (5), 46–49 (2006)
13. He, H., Qu, Z.: Design pattern mix structure method research and application. Computer Engineering and Design (3), 999–1001 (2007)
14. Feng, S., Hou, H.: Analysis Method and Evaluation of the Results for Object Oriented Software Metrics. Computer Engineering (7), 41–43 (2006)
15. Duan, Y., Zhang, S., Huang, H.: Improving on object oriented software are coupling metric. Computer Applications and Software (3), 6–8 (2007)