Journal of Intelligent & Fuzzy Systems 23 (2012) 53–60
DOI:10.3233/IFS-2012-0495
IOS Press

A fuzzy matching approach for design pattern mining

Yuxin Wang[a,*], He Guo[a], Hongbo Liu[a] and Ajith Abraham[b]
[a]*School of Computer, Dalian University of Technology, Dalian, China*
[b]*Machine Intelligence Research Labs – MIR Labs, USA IT For Innovations – EU Center of excellence, VSB – Technical University of Ostrava, Ostrava, Czech Republic*

Abstract. Mining design patterns from source code is significant for improving the intelligibility and maintainability of software. In this paper, we present a new design pattern matching method based on fuzzy, in which matrix model is used for describing both design pattern and source code, and design pattern's static and dynamic information is defined as fuzzy attribute value for measuring the matching degree. Experiments on three open-source projects demonstrate the accuracy and efficiency of the proposed methodology.

Keywords: Design pattern, fuzzy matching, pattern mining, matrix model

1. Introduction

Design patterns have been widely adopted by software industry and attracted significant attention in the last period. The main reason behind this is that design patterns are obviously useful in both development of new system and comprehension of existing object-oriented design. With the increasing of software system's scale and complexity, especially for some large legacy systems lacking of software architecture and design documentation, mining design patterns from source code can greatly improve the systems' maintainability and understandability.

The essence process of design pattern mining is pattern matching, hence the method chosen for matching directly influences the quality of mining. In consideration of the fuzzy and subjective characteristics of design pattern description, learning from human's fuzzy decision-making ability, we apply fuzzy method to pattern matching strategy and propose a practical approach for design pattern mining.

The rest of the paper is organized as follows. In the following section, we discuss some related work on design pattern mining. In Section 3, we describe the proposed methodology of mining in detail. Section 4 presents a case study to demonstrate the complete mining process. Section 5 shows and analyses the experiment results of our methodology and illustrates an example to prove its advantages. Finally, conclusions are given in Section 6.

2. Related work

With the development of software engineering, the technology of design pattern mining has drawn more and more concerns. A number of techniques have been proposed to mine design patterns from source code. Balanyi and Ferenc introduce a method to recover design patterns using a XML-based language [2] , called

*Corresponding author. Yuxin Wang, School of Computer, Dalian University of Technology, China. E-mail: wyx@dlut.edu.cn.

Design Pattern Markup Language (DPML). Kaczor et al. propose an efficient Iterative Bit-vector algorithm for design pattern identification [11]. In [8] an approach based on fingerprint is proposed to detect design pattern using metrics and machine learning techniques. Jing Dong et al. present an approach based on matrix to discover design patterns, in which matrix is used to describe the structural characteristics of the design patterns, and the value of each cell represents the relations among the classes [3].

Heuzeroth [9] and Li [12] think that neither static nor dynamic analysis alone is sufficient. So they propose detection methods combining static and dynamic information to detect design patterns. H. Huang et al. also consider both structural and behavioral aspect of design patterns important for improving the precision of patterns mining [10]. Their approach introduces predicate logic combined with interval-based temporal logic to convert the design patterns and source code into Prolog representations.

An existing similarity score algorithm is applied in [14] to detect design patterns, in which relationships among classes and all pattern characteristics are represented by matrix. The design pattern detection is achieved by calculating the similarity matrices using similarity score algorithm. The limitation of this approach is that the algorithm can only calculate the similarity between two vertices, instead of two graphs. High similarity score of two vertices does not guarantee a match between two sets of vertices. Aiming at these limitation, Jing Dong et al. present an optimized approach called normalized Cross Correlation (*CCn*) based on the template matching method [4]. Design pattern and system information are encoded into a single matrix using prime numbers. And the *CCn* approach calculates the similarity between each pair of vectors which transform from matrix. This approach is effective, but the dimension of pattern matrix must be same as the one of source code matrix, which would cause more complexity to some extent.

Research and technological developments of design pattern mining encourage the producing of many mining tools (e.g.,[6, 13, 16]) and many effective strategyies (e.g., Antoniol et al. proposed a multi-stage reduction strategy to extract structural design patterns from system, using metrics, structural properties and delegation constraints technique to reduce the exploration space [1]. M. Vokac used reverse engineering tool to analyze the source code and produce the structural information, which is stored in database, and the mining strategy is through database queries [15]).

In this paper, a design pattern mining method based on fuzzy theory is proposed. Matrix model based on prime number is introduced to describe the source code and design pattern. Clustering analysis is applied to optimize the source code matrix, which improves the efficiency of pattern matching. On the basis of fuzzy concepts, we take the static and dynamic information as the attributes so as to improve the matching accuracy. Our proposed approach is evaluated on three open source projects JUnit, JHotDraw and JFactory. The experiment results on different systems illustrate that this approach makes some progress in accuracy and integrity. It also can identify better the special design patterns which have the same structure but different intention.

3. Design pattern matching based on fuzzy method

As the essence of design pattern mining, pattern matching, which is subjective and fuzzy due to the descriptive characteristics of design pattern, is crucial for the mining process. So the fuzzy set theory and method could be introduced to apply to pattern matching process.

Fuzzy set theory, which is initiated by Zadeh, provides a framework for handling the uncertainties [17]. In non-fuzzy set every object is either a member of the set or not, but in fuzzy sets every object is to some extent member of a set and to some extent of another set. Thus, unlike the crisp sets membership is a continuous concept in fuzzy sets. Fuzzy is used in support of linguistic variables and there is uncertainness in the problem. Fuzzy theory and method are widely applicable in information gathering, modeling, analysis, optimization, control, decision making and supervision [5].

In this section, we present a fuzzy matching method to improve the integrality and accuracy of design pattern mining.

3.1. Fuzzy matching algorithm

Before we present the fuzzy matching algorithm, the relevant basic concepts are defined. Our fuzzy matching algorithm is illustrated in Algorithm 1.

Definition 1. *(Fuzzy set)* If X is a collection of objects denoted generically by x then a fuzzy set A in X is a

Algorithm 1. Fuzzy Matching Algorithm

Step 1. Build the models of both design pattern and source code;
Step 2. Define an attribute set $X = \{x_1, x_2, \ldots x_r\}$ for source code mode:
The main information in design pattern includes the class information, the static structure information and the dynamic behavior information. So the attribute set is defined as $X = \{X_{ClassInfo}, X_{StructureInfo}, X_{BehaviorInfo}\}$;
Step 3. Compute the attribute's matching degree vector:
Aiming at the design patterns which will be detected, compute the vector $T = \{T_{ClassInfo}, T_{StructureInfo}, T_{BehaviorInfo}\}$ of each source code model;
Step 3. Assign weight vector:
Give the weight vector $W = \{W_{ClassInfo}, W_{StructureInfo}, W_{BehaviorInfo}\}$ for each attribute according to their importance;
Step 4. Make Decision:
If $GMD(T_i \wedge W_i) \geq$ the threshold of certain design pattern, source code model i is the instance of this design pattern.

set of ordered pairs: $A = \left\{(x, \mu_A(x))|x \in X\right\}$. $\mu_A(x)$ is called the membership function or grade of membership of x in A which maps X to the membership space M.

Definition 2. *(Fuzzy relation)* Let $X, Y \subseteq R$ be universal sets then $R = \left\{\left((x, y), \mu_R(x, y)\right)|(x, y) \subseteq X \times Y\right\}$ is called a fuzzy relation on $X \times Y$.

Definition 3. *(Composition of fuzzy sets)* Let $Q \in F(U \times V)$, $R \in F(V \times W)$ be two fuzzy relations. The composition $Q \circ R$ (Q max $-$ min R) is the fuzzy set:
$\mu_{Q\circ R}(u, w) = \left\{ \bigvee_{v\in V}(\mu_Q(u, v) \wedge \mu_R(v, w))|u \in U, v \in V, w \in W\right\}$.

It is again the membership function of a fuzzy relation on fuzzy sets.

Definition 4. *(Maximizing set of two fuzzy sets)* Let A and B be two fuzzy sets. Maximizing set of A and B is a fuzzy set M that consists all supports from A and B. Membership degree of each support equals the ratio of the support itself to the maximum support of A and B.

Definition 5. *(Greatest Membership Degree of a fuzzy set)* Let A be a fuzzy set. $GMD(A)$ presents the greatest membership degree of A. $GMD(A) = \mu_A(x_i) = \max(\mu_A(x_1), \mu_A(x_2), \ldots \mu_A(x_n))$.

3.2. Matrix model of design pattern

Currently, design patterns are usually described as some half-formalized styles. In order to mine design patterns from source code, the formalized descriptive method for design pattern need be considered. On the basis of UML and natural language, we use the matrix based on prime number to describe the pattern.

Table 1
Prime Numbers of relationships between classes

Relations	Prime number value
Generalization	2
Association	3
Dependency	5
Realization	7
Aggregation	11
Composite	13

According to different intentions, design patterns are divided into three categories by GOF: Creational, Structural and Behavioral [7]. No matter what category the design patterns belong to, their structures can be represented by the relations between classes, which are very suitable for being described by matrix [3, 11]. Due to the complexity of these relations, the value of matrix cell should have the ability to be identified clearly. Realizing that the product of prime numbers is always a unique composite number [3], we represent relationships by different prime numbers and associate the most frequently used relation with the lowest prime number. The prime number representation of different relations between classes is shown in Table1.

Based on above representing method, the value of matrix element can be calculated as Equation 1.

$$M(i, j) = \begin{cases} \prod_{n=1}^{N} V_n(i, j) & N \geq 1 \\ 0 & N = 0 \end{cases} \quad (1)$$

Where N is the number of relations between class i and class j. $V(i, j)$ is the prime number value of a certain relation between class i and class j.

According to the method, we transform the 23 common design patterns proposed by Gamma et al. into matrices and put them into a maintainable design pattern model library.

3.3. Establishment of source code model

It is difficult to mine design pattern from the source code directly, so we need a kind of intermediate representation for the source code. The choice of this representation method will directly affect the design of matching algorithm. In order to get an exact match, matrix defined for design pattern in previous section is still used to describe source code. The transformation flow from source code to matrix is shown in Fig. 1.

Sparx Enterprise Architect (EA) is a famous UML tool. Firstly, we use the function of generation and reverse engineering of EA to get the UML diagram of source code and then export the model to XMI (XML Metadata Interchange) file; Secondly, extract useful information for mining from the XMI file by XSLT and generate the XML file; Lastly convert the XML to matrix.

According to the flow, source code can be transformed to a n by n square matrix (n is the number of classes of the source code). Evidently, when a certain design pattern is needed to be mined, the overall matrix of source code must contain lots of redundancy information, which will seriously influence the efficiency of matching. Hence, the source code matrix should be optimized before mining, for example, by a Model-based Clustering Method. Design pattern matrices are set as the referenced models and the optimal matches are extracted from source code matrix.

3.4. Computation of source code model's matching degree

For the matching degree vector $T = \{T_{ClassInfo}, T_{StructureInfo}, T_{BehaviorInfo}\}$, each attribute in it can be calculated as follows.

3.4.1. Class information's matching degree

A class's information includes its method, property and type. For the process of design pattern mining only the class's type is concerned about, and the type could be interface, abstract or normal. Based on the characteristics of class information and our experiences, a quadratic parabolic membership function is defined for $T_{ClassInfo}$ as Equation 2.

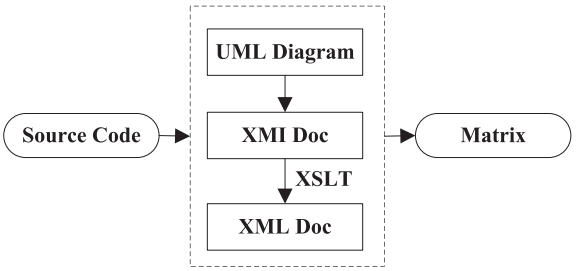

Fig. 1. The Flow from Source Code to Matrix.

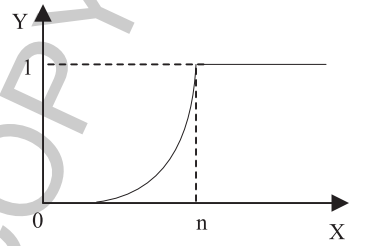

Fig. 2. Value Distribution of $T_{ClassInfo}$.

$$T_{ClassInfo} = \begin{cases} 0 & x \le 1 \\ \left(\frac{x-1}{n-x}\right)^2 & 1 < x < n \\ 1 & x \ge n \end{cases} \quad (2)$$

Where the cutoff point is denoted by n which is the number of a design pattern's abstract classes; x is the number of abstract classes of some candidate source code instance. So $T_{ClassInfo}$ is a number between 0 and 1, and the larger the value is the higher degree of similarity between the candidate instance and the design pattern. The value distribution of $T_{ClassInfo}$ can be described as Fig. 2.

3.4.2. Structure information's matching degree

In [4], normalized cross correlation (CCn) template matching method is introduced by dividing the cross correlation value by the product of the norms of vector f and g as Equation 3, which is really effective for matching matrix models.

$$CCn = \frac{\sum f(x) \cdot g(x)}{|f(x)| \cdot |g(x)|} \quad (3)$$

In other words, CCn defines the $cos\theta$ value, where θ is the angle between f and g. The maximum value is 1 when f and g is an exact match, i.e., $\theta = 0$.

On the basis of the optimization of source code matrix, we apply *CCn* method to match the static structure information of design pattern model and source code model. The matching result is obtained. The value is between 0 and 1, where 1 means exact match and 0 means no match. It is the matching degree of structure information $T_{StructureInfo}$.

3.4.3. Behavior information's matching degree

Many mining approaches can not distinguish the design patterns with same structure but different intention (e.g., Composite vs. Decorator and State vs. Strategy). The same static models lead to a matched detection result. In order to overcome the limitation, we introduce the dynamic behaviour information as a mining factor.

Dynamic behavior information's matching degree can be acquired through the following steps:

1. Get the behavior information of both design pattern and source code from the corresponding sequence diagram;
2. Transform the sequence diagram to behavior matrix:
   The matrix is defined as: column represents the involved classes; row represents the detailed behavior (row i is the task i). The value of matrix cell can be positive, negative and ∞, where positive is the begin of the task, negative is the end of the task, ∞ represents the periods is finished;
3. Compute the similarity of the dynamic matrices:
   Use *CCn* to calculate the similarity of the matrices and the result is the dynamic information's matching degree value.

Figure 3 is the State pattern's sequence diagram, and its dynamic behavior matrix is $\begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 2 & -2 & 0 \\ 0 & -3 & 3 & 0 \\ 4 & -4 & 0 & 0 \\ 0 & 5 & 0 & -5 \\ 0 & -6 & 0 & 6 \\ \infty & \infty & \infty & \infty \end{bmatrix}$.

4. Case study

In this section, a case study is provided to demonstrate the complete process mentioned above for mining Visitor pattern.

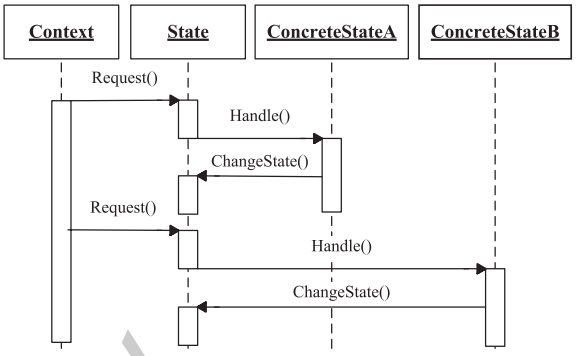

Fig. 3. State Pattern's Sequence Diagram.

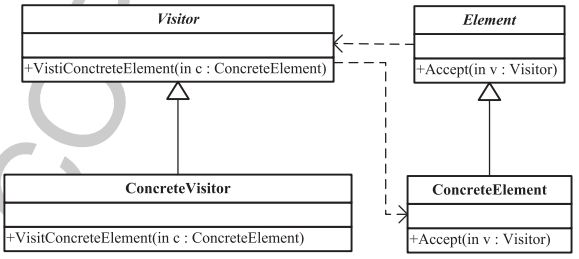

Fig. 4. Visitor Pattern's UML Diagram.

1. Matrix Model of Visitor Design Pattern
   Firstly we get the prime number matrix model of Visitor pattern whose UML diagram is showed in Fig. 4 as $\begin{bmatrix} 0 & 0 & 0 & 5 \\ 5 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$.
2. Matrix Model of Source Code and its Optimization
   Figure 5 shows the source code's UML diagram. According to the transformation flow from source code to matrix, we can get the source code's matrix mode is a 8×8 matrix.
   Evidently, in order to mine Visitor pattern, there are a lot of redundancy information. So, we need optimize the original matrix according Model-based Clustering Method. And the clustering condition has to be the *Generalization* and *Dependency* relations. After clustering, we can get two matrices which are $\begin{bmatrix} 0 & 0 & 0 & 5 & 5 \\ 5 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix}$ and

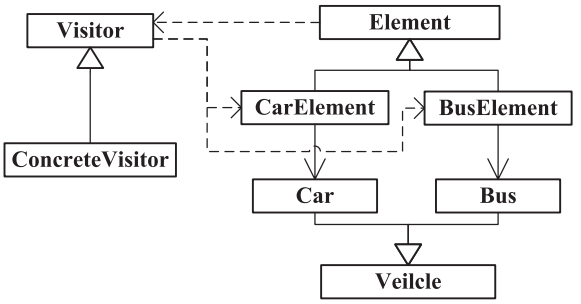

Fig. 5. UML Diagram of Partial Source Code.

$\begin{bmatrix} 0&0&0 \\ 2&0&0 \\ 2&0&0 \end{bmatrix}$. However, there is still some reduplicate information in the matrices, such as the last 2 rows and columns are same in first matrix, that means class *CarElement* and *BusElement* have the same signification, which needs to be normalized, and so does the second matrix. After that we can get $\begin{bmatrix} 0&0&0&5 \\ 5&0&0&0 \\ 2&0&0&0 \\ 0&2&0&0 \end{bmatrix}$ and $\begin{bmatrix} 0&0 \\ 2&0 \end{bmatrix}$. Furthermore, since the dimension of second matrix is less than Visitor pattern's, the matrix must not be the instance of Visitor Pattern. So we get the optimized matrix $\begin{bmatrix} 0&0&0&5 \\ 5&0&0&0 \\ 2&0&0&0 \\ 0&2&0&0 \end{bmatrix}$. The final clustering result shows that there is almost no redundancy information and the number of matching decrease from C_8^4 to 1, the efficiency of matching is greatly improved.

3. Pattern Matching

Through above analysis, we get the matrix models of design pattern and source code. And next, we will compute the Matching Degree of the two models. According to Equation (2), the matching degree of class's information is 1. Also, the structure information's matching degree is 1 which is calculated by *CCn* method.

As discussed in section 3, we get the behaviour matrix of Visitor pattern and source code as $\begin{bmatrix} 1&-1&0&0 \\ 0&2&0&-2 \\ 0&-3&0&3 \\ 4&0&-4&0 \\ 0&0&5&-5 \\ 0&0&-6&6 \end{bmatrix}$. So the matching degree of behaviour information is also 1.

According the matching degree, the matching degree vector of Visitor pattern is $T_{Visitor} = (1.0, 1.0, 1.0)$. By defined the weight vector of Visitor pattern as $W_{Visitor} = (0.4, 0.83, 0.65)$, we can get $GMD(T_{Visitor} \wedge W_{Visitor}) = 0.83$ which is greater than Visitor pattern's threshold 0.73. So, the mining result is that the source code includes one candidate of Visitor pattern.

5. Analysis of experiment results

Our proposed approach is evaluated on three open source projects JUnit, JHotDraw and JFactory. There are a lot of well-known design patterns in them, and the authors indicate the implemented design patterns in the documentation. Moreover, many related works take them as the specification of design pattern mining, and we can compare their results with ours. Table 2 illustrates part of the mining results of behavioral patterns by our approach (M_O) and compares them with method of [8] (M_G) and method of [4] (M_D).

Table 2 illustrates that our approach can distinguish State pattern from Strategy pattern well and the mining accuracy and integrity is improved.

We take the source code of JHotDraw (shown in Fig. 6) as an example to analyze the effectiveness of our

Table 2
Performance comparison of considered approaches

Patterns	JUnit			JHotDraw			JRefactory		
	M_O	M_G	M_D	M_O	M_G	M_D	M_O	M_G	M_D
Version	v4.7	v3.7	v3.8	v6.0	v5.1	v6.0	v2.9.18	v2.6.24	v2.6.24
State (strategy)	0 (3)	3	3	12 (17)	22	29	7 (15)	11	12
Template method	1	1	–	5	5	–	19	17	–
Observer	4	4	–	3	5	–	1	0	–
Visitor	0	0	–	1	1	–	2	2	–

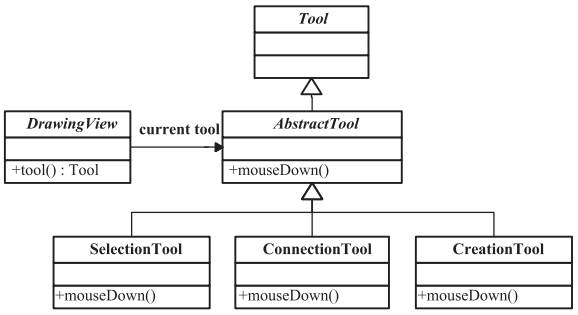

Fig. 6. Part of Source Code of JHotDraw.

approach. It is an illustration of State pattern. According to [4], the mining result is that the source code is the instance of both State pattern and Strategy pattern, because these two patterns have the same static structure, i.e., they have the same matrix and the same matching result. That means they can not distinguish State pattern from Strategy pattern.

In our approach, the matrices of state and strategy pattern both are $\begin{bmatrix} 0 & 3 & 0 \\ 0 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$, and $\begin{bmatrix} 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{bmatrix}$ is the optimized source code matrix. Then $T_{ClassInfo\_State} = 1.0$, $T_{StructureInfo\_State} = 0.97$, $T_{BehaviorInfo\_State} = 0.81$, so the source code matching degree vector of state pattern is $T_{State} = (1.0, 0.97, 0.81)$. Here only one cycle's behavioral information is considered, we need not to consider the whole life cycle of the source code sequence diagram, so the matrix is $\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 2 & -2 & 0 & 0 \\ 0 & -3 & 3 & 0 & 0 \\ 4 & -4 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & -5 \\ 0 & -6 & 0 & 0 & 6 \\ 7 & -7 & 0 & 0 & 0 \\ 0 & 8 & 0 & -8 & 0 \\ 0 & -9 & 0 & 9 & 0 \end{bmatrix}$. Similarly, $T_{Strategy} = (1.0, 0.97, 0.23)$.

Define the weight vector of these two patterns as $W_{State} = (0.3, 0.8, 0.7)$, $W_{Strategy} = (0.3, 0.7, 0.6)$, and according to our experience, State pattern's threshold is 0.75 and Strategy pattern's is 0.79. Then we can get $GMD(T_{State} \wedge W_{State}) = 0.81 > 0.75$ and $GMD(T_{Strategy} \wedge W_{Strategy}) = 0.7 < 0.79$, so the result illustrates that the source code is an instance of state pattern.

6. Conclusion

In this paper, an approach of design pattern matching based on fuzzy theory is proposed to implement design pattern mining. The description of both source code and design pattern is formalized through prime-number-based matrix model. Clustering analysis is applied to optimize the source code matrix. The design pattern's static and dynamic information is defined as the attributes for measuring the matching degree. And finally the fuzzy matching between design pattern and source code is implemented based on fuzzy theory. Experiment results illustrate that our approach can detect design patterns from the source code well, especially for the behavioral design patterns with higher accuracy and efficiency.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 60873054, 61073056), the Fundamental Research Funds for the Central Universities (Grant No. 2011JC006), the Science and Technology Research Project of Education Bureau of Liaoning Province (Grant No. 2009S013), and the Dalian Science and Technology Fund (Grant No. 2010J21DW006).

References

[1] G. Antoniol, G. Casazza, M.D. Penta and R. Fiutem, Object oriented design pattern recovery, *Journal of Systems and Software* **59**(2) (2001), 181–196.

[2] Z. Balanyi and R. Ferenc Mining design patterns from C++ source code, *Proceedings of the 19th IEEE International Conference on Software Maintenance*. Amsterdam, The Netherlands, 2003, 305–314.

[3] J. Dong, D. Lad and Y. Zhao, DP-Miner: Design pattern discovery using matrix, *Proceedings of 14th annual IEEE International Conference and Workshops the Engineering of Computer-Based Systems (ECBS)*. Arizona, USA, 2007, 371–380.

[4] J. Dong, Y. Sun and Y. Zhao, Design pattern detection by template matching, *Proceedings of the 23rd Annual ACM Symposium on Applied Computing*. Fortaleza, Cear, Brazil, 2008, 765–769.

[5] M. Fasanghari and F.H. Roudsari, Optimized ICT project selection utilizing fuzzy system, *World Applied Sciences Journal* **4**(1) (2008), 44–49.

[6] R. Ferenc, J. Gustafsson, L. Miiller and J. Paki, Recognizing design patterns in C++ programs with the integration of Columbus and maisa, *Acta Cybernetica* **15**(4) (2002), 669–682.

[7] E. Gamma, R. Helm and R. Johnson, et al., Design patterns: *Elements of reusable object-oriented software*, Addison Wesley Publishing Company, 1995, 21–22.

[8] Y. Guéhéneuc, H. Sahraoui and F. Zaidi, Fingerprinting design patterns, *Proceedings of the 11th Working Conference on Reverse Engineering*. Delft, The Netherlands, 2004, 172–181.

[9] D. Heuzeroth, T. Holl, G. Hogstrom, and W. Lowe, Automatic design pattern detection, *Proceedings of the 11th International Workshop on Program Comprehension (IWPC)*. Portland, Oregon, USA, 2003, 94–103.

[10] H. Huang, S. Zhang, J. Cao and Y. Duan, A practical pattern recovery approach based on both structural and behavioral analysis, *Journal of Systems and Software* **75**(1-2) (2005), 69–87.

[11] O. Kaczor, Y. Guéhéneuc and S. Hamel, Efficient identification of design patterns with bit-vector algorithm, *Proceedings of the Conference on Software Maintenance and Reengineering*. Bari, Italy, 2006 175–184.

[12] F. Li, Q. Li, Y. Su and P. Chen, Detection of design patterns by combining static and dynamic analyses, *Journal of Shanghai University* **11**(2) (2007), 156–162.

[13] L. Prechelt and C. Krämer, Functionality versus practicality: Employing existing tools for recovering structural design patterns, *Journal of Universal Computer Science* **4**(12) (1998), 866–882.

[14] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides and S. Halkidis, Design pattern detection using similarity scoring, *IEEE Transaction on Software Engineering* **32**(11) (2006), 896–909.

[15] M. Vokac, Defect frequency and design patterns: An empirical study of industrial code, *IEEE Transactions on Software Engineering*, **30**(12) (2004), 904–917.

[16] M. Vokac, An efficient tool for recovering design patterns from C++ code, *Journal of Object Technology* **5**(1) (2006), 139–157.

[17] L.A. Zadeh, Fuzzy sets, *Information and Control* **8** (1965), 338–353.