International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 11, November-2019

A Review on Design Pattern Detection

Rajwant Singh Rao
Department of Computer Science & Information Technology
Guru Ghasidas Vishwavidyalaya (A Central University),
Bilaspur (C. G.)

***Abstract*— In many object oriented software, there are recurring patterns of classes. With the help of these patterns specific design problem can be solved and object oriented design become more flexible and reusable. Design Pattern existence improve the program understanding and software maintenance. Hence a reliable design pattern mining is required. Graph Matching algorithms are useful and very general form of pattern matching to find the realistic use in several areas. In this paper we are reviewing the different graph matching algorithms to detect design patterns.**

Keywords*— Design pattern, UML, matrix, subgraph isomorphism

I. INTRODUCTION

To reuse expert design experiences, the design patterns [1] have been extensively used by software industry. When patterns are implemented in a system, the pattern-related information is generally no longer available. To understand the systems and to modifications in them it is necessary to recover pattern instances. For graph matching there are many algorithms, we are reviewing some of the algorithms which can be applied for design pattern detection. The details of each of the algorithms are discussed in sections.

I. Exact graph matching algorithms

Exact graph matching algorithms [7], used to find a one-to-one mapping (isomorphism) between the nodes of two graphs which have the same number of nodes so that there is also a one-to one mapping between the related edges. In the context of design pattern detection, the application of such an algorithm would require the Examination of all possible sub graphs of the system Graph that have the same number of vertices with the pattern. The most important drawback, is that a given design pattern may be implemented in various forms that differ from the basic structure found in the literature, and as a result exact matching is insufficient for design pattern detection.

II. Inexact Graph Matching Algorithm

The inexact graph matching procedure is used when an one to one correspondence is not found between two graphs and the purpose is to find the finest corresponding between both graphs. There are procedures which compute the edit distance between two graphs [1], defined as the number of alterations required to reach from one graph to the other graph. In the background of design pattern finding this gives incorrect results.

In inexact graph matching process which graph takings fewer number of alterations to reach to the goal graph is supposed to the nearer to the goal graph.

III. Kleinberg Approach for Vertices Scoring

Kleinberg [3] offered a link analysis system for identifying pages on the web (by computing hub and authority). That is authoritative sources on broad search queries.

The hubs can be defined as pages which points to several good authorities and authorities are the pages which are pointed to by several good hubs. The disadvantage of this process is that it only computes the similarity between two nodes rather than whole graph. To remove this drawback another approach is introduced, called sub graph isomorphism detection.

IV. Exact and Approximate Matches

There is lack of research on the set of characteristics of each pattern. It needs to be checked, most of the researchers simplify the problem by making their own definitions. Most approaches give a piece of architectural design which structurally confirms to the structural characteristics of the pattern that they defined and behaviorally exhibits the expected actions that they defined. As the time over they find that all matching rules are conformed and they claim to find a match. If some of the rules are not conformed they claim a mismatch. Therefore there is a need to solve this problem. But different definition of the same pattern seems to make different search results. Some pattern candidates which are real instances are filtered out due to strict exact matches and some approaches perform approximate matches by computing the similarity degree.

V (a). Similarity Scoring

V (c) Comparative study between exact match and approximate matching

Exact matching is required when the system piece is exactly the same as the pattern, i. e., a range in [0, 1] can be used as the matching degree So the exact matching selects system pieces with a matching degree of 1, and eliminates those with a degree less than 1. Exact matching approaches usually omit the effort of calculating matching degrees. While approximate matching sets a number between 0 and 1 as the threshold. For example 0.8 can be a reasonable threshold. Thus, any system piece with a matching degree equal to or higher than 0.8 is retained and others are filtered out when matching a particular pattern.

Niere *et al.* [47] purposed fuzzy-belief, a value between 0 and 1, for each structural rule which express its precision. To limit the rule applications to reasonable cases, they introduce thresholds to the rules. The match with a fuzzy value lower

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 11, November-2019

than the threshold will be excluded. This helps to minimize computation load and improve the scalability. Tsantalis *et al*. [44] applied a similarity scoring algorithm, an inexact graph matching algorithm, to compute matching degree between system under study and the pattern, when an exact isomorphism between two graphs cannot be found. Dong *et al*. [6] applied a template matching algorithm to calculate the matching degree between a piece of system and a design pattern. Guéhéneuc *et al.* [42] propose a machine learning algorithm to compute software metrics and use the degree of confidence to infer the rules by a rule learner. Ferenc *et al*. [43] introduce predictors for each pattern and use machine learning algorithm training the pattern recognizer to acquire the value of the predictors. These predictor values are then used as the standards to compare with the predictor values 13 obtained for system under study. A pre-processing algorithm is also proposed by Dong *et al.* [48] for generating the training set of machine learning algorithms for pattern mining. Table 1 gives the categorization of different tools on exact or approximate matching.

Matching Technique	Authors	Tools
Exact Match	Kramer 1996 [12]	Pat
	Seemann 1998[19]	
	Bansiya 1998[8]	DP++
	Antoniol 1998 [14]	
	Tonella 1999 [20]	
	Keller 1999 [10]	SPOOL
	Blewitt 2001 [21]	Hedgehog
	Mei 2001 [22]	JBOORET
	Albin-Amiot 2001 [23]	Ptidej
	Asencio 2002 [24]	
	Wendehals 2003 [25]	
	Smith 2003 [26]	SPQR
	Heuzeroth 2003 [27][28]	
	Beyer 2003 [29]	CroCoPat
	Park 2004 [30]	
	Zhang 2004 [31]	
	Costagliola 2005 [32][33]	DPRE
	Streitferdt 2005 [34]	
	Huang 2005 [35]	PRAssistor
	Wang 2005 [36]	DPVK
	Kaczor 2006 [37]	Ptidej
	Shi 2006 [38]	PINOT
	Dong 2007 [39][40]	DP-Miner
Approximate Match	Niere 2002 [41]	FUJABA
	Guéhéneuc 2004 [42]	
	Ferenc 2005 [43]	Columbus
	Tsantalis 2006 [44]	

Table 1 Categorization of Current Discovery Method Based on Matching Techniques [48]

VI. Sub Graph Isomorphism Detection

A. Theoretical Approach

The subgraph isomorphism is a convenient generalization of graph isomorphism. The subgraph isomorphism problem [4] is define whether a graph is isomorphic to a subgraph of any other graph. Consider [5] G1 (V1, E1) and G2 (V2, E2) be two graphs, where V1, V2 are the set of nodes and E1, E2 are the set of edges. Let M1 and M2 be the adjacency matrices of dimensions m x m and n x n, where m < =n, corresponding to the graphs G1 and G2 respectively. A permutation matrix is a square (0, 1)-matrix that has exactly one entry 1 in each row and each column and 0's elsewhere. Two graphs G1 (M1, Lv, Le) and G2 (M2, Lv, Le) are said to be isomorphic [5] if there exist a permutation matrix P such that

$$M2 = P\ M1\ P^T \qquad (1)$$

A subgraph *S* of a graph *G*, $S \subseteq G$, is a graph $S = (M^i, L_v, L_e)$ where $M^i = S_{m,m}(P\ M\ P^T)$ is an m x m adjacency matrix for some permutation matrix P.

There is a subgraph isomorphism from G1 to G2 if there exists an n x n permutation matrix P such that

$$M1 = S_{m,m}(P\ M2\ P^T) \qquad (2)$$

We take M2 matrix as a system design matrix and we guess nondeterministically M1 as a design pattern matrix. And then try to find out whether M1 is subisomorphic to M2 or not or it can be easily said whether there exist design pattern in the system graph or not.

Hence the problem of finding a subgraph isomorphism from graph G1 to G2 is equivalent to finding a permutation matrix for which equation (2) holds. Thus, we generate permutation adjacency matrix of a model graph (system under study) one by one and check whether equation (2) holds or not, when it holds we stop and declare that that particular design pattern has been detected. It can be also possible that there is no design pattern exists in system graph. In this case we find no permutation matrix for which equation (2) holds.

VII. Searching for minimal key structures

In this method a defined key structure is associated to each pattern. The key structure for pattern describes the number of minimum classes and objects that are present in that structure. By define the key structure of pattern the pattern can be securely identified. The properties of key structure are used as the search criteria. There are three software systems for automated searching is known which are based on this approach. These are DP++[8], for C++, KT [9] for Smalltalk and SPOOL [10] realized for C++, applicable for Java and Smalltalk.

The DP++[8] searches the following design patterns: COMPOSITE, DECORATOR, ADAPTER, FACADE, BRIDGE, FLYWEIGHT, TEMPLATE METHOD and CHAIN OF RESPONSIBILITY. DP++ [8] is not applicable for other patterns. This tool has three parts: (i) C++ Code Translation Subsystem used for analyzing source code. (ii) Pattern Detection Subsystem used for the recognition of generation patterns and (iii) Display Subsystem used for the visualization of detected patterns. Information about the achieved values of recall and precision is not available.

KT [9] is able to search COMPOSITE, DECORATOR, STATE, STRATEGY, COMMAND, TEMPLATE METHOD and CHAIN OF RESPONSIBILITY patterns. It is unable to recognize INTERPRETER pattern .The information about other patterns are not mentioned.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 11, November-2019

Information about the achieved values of recall and precision is not available.

SPOOL [10] is capable of searching BRIDGE, FACTORYMETHOD and TEMPLATEMETHOD patterns.

VIII. Searching for class structures

This approach uses the pattern class structures described by Gamma patterns [11].For example consider the following figure 18. This is the composite pattern. A composition pattern exit if a class has at least two subclasses and one of them has 1 to n aggregation to the super class. There are three software systems for automated search which are based on this approach. These are Pat [12] for C++, IDEA [13] for UML diagrams and the multi-step search tool [14].

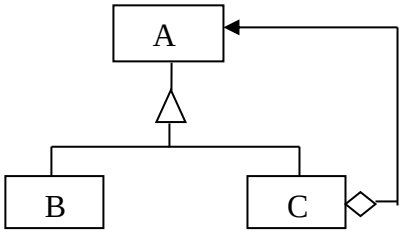

Fig.18. A Composite Pattern

patterns, such as Kramer [12] and Costagliola [32][33], usually discover structural patterns with few behavioral patterns. Approaches which takes both structural and behavioral patterns, such as Heuzeroth *et al.* [27][28], are able to discover more behavioral patterns.

XII. Experiments

To evaluate the different approaches experiment is the good way. Table 4 shows a list of software system used in the experiments by different studies. It is difficult to evaluate the precision value of different approaches since there is a lack

Pat [12] describes the pattern class structure by PROLOG rules. This tool is able to finds ADAPTER, PROXY, BRIDGE, DECORATOR and COMPOSITE patterns. This search tool has been been tested with software systems containing 9–343 classes. The achieved recall value in each case is 100% and the average precision value is 36.75%. IDEA [13] based on UML search approach which uses class and collaboration diagrams. It also used PROLOG rules. This is able to find the following patterns: TEMPLATE METOD, PROXY, ADAPTER, BRIDGE, COMPOSITE, DECORATOR, FACTORY METHOD, ABSTRACT FACTORY, ITERATOR, OBSERVER and PROTOTYPE.

The multi step search tool [14] is able to find the following patterns ADAPTER, BRIDGE, PROXY, COMPOSITE and DECORATOR. This is unable to recognize the other patterns.

The multi step search tool was tested with different C++ Libraries [14]. The achieved recall value is 100% and an average precision value is 35%..

XI. Discovered Patterns

Since there are large numbers of available patterns, each approach often considers only a small number of patterns. The following table 3 shows the summarization of patterns discovered by different tools. This table show that the approaches that focus to the structural aspect of
of document of these systems, especially for open source system.

XII. Analysis of Experiment Results

As shown in Table 2 and Table 4 most approaches given experiments on mining different patterns from some application systems. Based on study in the previous section we found that different approaches give different results when mining the same pattern in the same system. For instance, Table 5 shows the comparison of the mining results of the same design patterns from the same systems by two different approaches.

Authors	Tools	Language	Structural (ST) Behavioral (BE) Semantic (SE)	Exact (EX) Approximate (AP) match	Autom-atic(AT) Inter-active(IT)	Techniques	System Representation	Pattern Representation
Kramer 1996 [12]	Pat	C++	ST	EX	AT	Prolog	Prolog	Prolog
Seemann 1998[19]		Java	ST	EX	AT	first order logic, graph	Graph	Predicate
Bansiya 1998[8]	DP++	C++	ST	EX	AT		class hierarchy	text
Antoniol 1998 [14]		C++	ST & BE	EX	AT	metrics	AST	AOL
Tonella 1999 [20]		C++	ST & BE	EX	AT	concept analysis		
Keller 1999 [10]	SPOOL	C++		EX	IT			UML/CDIF
Blewitt 2001 [21]	Hedgeh og	Java	ST & BE & SE	EX	AT			Spine
Mei 2001 [22]	JBOOR ET	C++	ST	EX				
Albin-Amiot 2001 [23]	Ptidej	Java		EX	AT		CSP	
Asencio 2002 [24]		C++	ST	EX	IT			
Wendehals 2003 [25]		Java	ST & BE	EX	AT	dynamic runtime data	ASG & call graph	
Smith 2003 [26]	SPQR	C++	ST	EX	AT	OTTER rho-calculus	OML & OTTER	otter rules
Heuzeroth 2003 [27][28]		Java	ST & BE	EX	AT	SanD and SanD-Prolog	AST	AST &TLA

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 11, November-2019

Beyer 2003 [29]	CroCoPat	Java	ST	EX	AT	predicate calculus	BDDs	predicates
Park 2004 [30]			BE	EX	AT		Class Diagram	
Zhang 2004 [31]			ST	EX	AT		Graph (matrix)	Graph (matrix)
Costagliola 2005 [32][33]	DPRE	C++ Java	ST	EX	AT	XPG formalism & LRbased parsing	SVG & AOL->AST	Grammar-based Pattern Specification
Streitferdt 2005 [34]		Java	ST	EX	AT			
Huang 2005 35]	PRAssistor			EX	AT			
Wang 2005 [36]	DPVK		ST & BE	EX	AT	REQL query	REQL static & RSF dynamic	REQL script
Kaczor 2006 [37]	Ptidej	Java	ST	EX	AT		bit representation	bit representation
Shi 2006 [38]	PINOT	Java	ST & BE	EX	AT	Data/Control Flows	AST	DFG & CFG
Dong 2007 [39][40]	DP-Miner	Java	ST & BE & SE	EX	AT	Matrix and Weight	XMI	XMI
Niere 2002 [41]	FUJABA	Java		AP	IT	bottom-up & top-down	ASG	ASG
Guéhéneuc 2004 [42]		Java	ST	AP	AT		XML tree & PADL	PADL
Ferenc 2005 [43]	Columbus	C++	ST	AP	AT		ASG, XML DOM tree	DPML
Tsantalis 2006 [44]		Java	ST	Ap	AT	Similarity Matrix	matrix	

Table 2 Comparison of Pattern Recovery Approaches

Authors	Abstract Factory	Adapter/Command	Builder	Bridge	Chain of Responsibilities	Command	Composite	Decorator	Facade	Factory Method	Flyweight	Mediator	Observer/MVC	Prototype	Proxy	Singleton	Strategy/State	Template Method	Visitor
Kramer 1996 [12]		×		×			×	×							×				
Seemann 1998[19]				×			×										×		
Antoniol 1998 [14]		×		×			×	×							×				
Keller 1999 [10]				×						×								×	
Blewitt 2001 [21]				×		×				×					×	×			
Asencio 2002 [24]	×			×				×		×					×	×	×		
Heuzeroth 2003 [27][28]					×		×	×				×	×						×
Balanyi 2003 [45]	×	×		×	×			×		×				×	×	×	×	×	×
Hericko 2005 [46]							×						×				×		
Costagliola 2005 [32] [33]		×		×			×	×							×				
Huang 2005 [35]	×	×	×	×	×		×	×		×	×	×	×		×	×	×		×
Kaczor 2006 [37]	×						×												
Shi 2006 [38]	×	×		×	×		×	×	×	×	×	×	×		×	×	×	×	×
Dong 2007 [39][40]		×		×			×										×		
Niere 2002 [41]				×			×										×		
Guéhéneuc 2004 [42]	×	×	×			×	×	×		×			×	×		×	×	×	×
Ferenc 2005 [43]		×															×		
Tsantalis 2006 [44]		×		×			×	×		×			×	×		×	×	×	×

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 11, November-2019

Table 3 Design Patterns Discovered by Different Approaches

Authors	Galib	LEDA	Libg++	Java AWT	Java Swing	JDK	JEdit	JHotDraw	JRefactory	JUnit	Mec	QuickUML	Socket	zApp class library	Other
Kramer 1996 [12]														×	×
Seemann 1998[19]				×											
Antoniol 1998 [14]	×	×	×								×		×		×
Tonella 1999 [20]											×				
Keller 1999 [10]															×
Blewitt 2001 [21]				×											
Albin-Amiot 2001 [23]				×			×	×		×					
Asencio 2002 [24]															×
Beyer 2003 [29]				×		×		×							×
Heuzeroth 2003 [27][28]					×										×
Balanyi 2003 [45]		×													×
Costagliola 2005 [32] [33]	×		×								×				×
Streitferdt 2005 [34]				×											×
Huang 2005 [35]								×		×					
Kaczor 2006 [37]								×			×				×
Shi 2006 [38]				×	×			×							×
Dong 2007 [39][40]				×			×	×		×					
Niere 2002 [41]				×											×
Guéhéneuc 2004 [42]								×	×	×		×			×
Ferenc 2005 [43]															×
Wang 2005 [36]															×
Tsantalis 2006 [44]								×	×	×					

Table 4 Experiments Done in Different Studies

Systems	JHotDraw5.1		JRefactory2.6.24		JUnit3.7	
Authors	Tsantalis *et al.* 2006 [44]	Guéhéneuc *et al.* 2004 [42]	Tsantalis *et al.* 2006 [44]	Guéhéneuc *et al.* 2004 [42]	Tsantalis *et al.* 2006 [44]	Guéhéneuc *et al.* 2004 [42]
Adapter	18	1	7	7	1	0
Composite	1	1	0	0	1	1
Decorator	3	1	1	0	1	1
Factory Method	3	3	4	1	0	0
Observer	5	2	0	0	4	3
Prototype	1	2	0	0	0	0
Singleton	2	2	12	2	0	2
State	23	2	12	2	3	0
Template Method	5	2	17	0	1	0
Visitor	1	0	2	2	0	0

Table 5 Different Results from the Same System of the Same Version

CONCLUSION:

The exact graph matching, inexact graph matching and Kleinberg Approach for vertices scoring shows the similarity between nodes not in the whole graph. This drawback, is reduced by sub graph isomorphism detection. The subgraph isomorphism shows whether a graph is isomorphic to a subgraph of any other graph or not .It uses the concept of Overall matrix to reduce the number of manipulations, it combines different matrices (like generalization matrix, association matrix, dependency matrix, aggregation matrix etc) into a single matrix. So there will be only two overall matrices, one corresponding to system and one for design pattern. After it we try to find out

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 11, November-2019

whether a particular design pattern exists on the given graph. This paper also shows different table on several studies.

REFERENCES:

[1] D.J. Cook and L.B. Holder, "Substructure Discovery Using Minimum Description Length and Background Knowledge,"J. Artificial Intelligence Research, vol. 1, pp. 231-255, Feb. 1994

[2] J.M. Smith, "An Elemental Design Pattern Catalog," echnical Report TR-02-040, Dept. of Computer Science, Univ. of North Carolina, Oct. 2002.

[3] J.M. Kleinberg, "Authoritative Sources in a Hyperlinked Environment," J. ACM, vol. 46, no. 5, pp. 604-632, Sept. 999.

[4] Gabriel Valiente, Algorithms On Trees And Graphs, 2002, page 367

[5] B.T. Messmer, H. Bunke, Subgraph isomorphism detection in polynomial time on preprocessed model graphs, Second Asian Conference on Computer Vision, 1995, pp. 151–155.

[6] Jing Dong, Yongtao Sun, Yajing Zhao, Design Pattern Detection By Template Matching, the Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC), pages 765-769,Ceará, Brazil, March 2008

[7] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T. Halkidis," Design Pattern Detection Using Similarity Scoring"

[8] Bansiya J (1998) Automatic Design-Pattern Identification. Dr. Dobb's Journal. Available online at: http://www.ddj.com

[9] Brown K (1996) Design reverse-engineering and automated design pattern detection in Smalltalk. Master's thesis. Department of Computer Engineering, North Carolina State University. Available online at: http://www.ncsu.edu/

[10] Keller RK, Schauer R, Robitaille S, Page P (1999) Pattern based reverse engineering of design components. In: Proc. Of the 21st International Conference On Software Engineering. IEEE Computer Society Press, pp 226–235

[11] Gamma E, Helm R, Johnson R, Vlissides J (1995) Design Patterns – Elements of Reusable Object-Oriented Software, Addison Wesley

[12] Kraemer C, Prechelt L (1996) Design recovery by auto-mated search for structural design patterns in object-oriented software. In: Proc. of the Working Conference on Reverse Engineering, pp. 208–215

[13] Bergenti F, Poggi A (2000) Improving UML design using automatic design pattern detection. In: Proc. 12th. International Conference on Software Engineering and Knowledge Engineering (SEKE 2000), pp 336–343

[14] Antoniol G, Fiutem R, Cristoforetti L (1998) Design pattern recovery in object-oriented software. In: 6th International Workshop on Program Comprehension, June, pp 153–160

[15] Niere J, Wadsack JP, Wendehals L (2001) Design pattern recovery based on source code analysis with fuzzy logic. Technical eport tr-ri-01-222, University of Paderborn. Available online

[16] Kim H, Boldyreff C (2000) A method to recover design patterns using software product metrics. In: Proc. of the 6th International Conference on Software Reuse (ICSR6).Available online at: http://www.soi.city.ac.uk/hkim69/ publications/icsr6.pdf

[17] V.D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. Van Dooren, "A Measure of Similarity between Graph Vertices: Applications to Synonym Extraction and Web Searching," SIAM Rev.,vol. 46, no. 4, pp. 647-666, 2004.

[18] D.H. Ballard and C.M. Brown, Computer Vision, Prentice Hall, 1982.

[19] J. Seemann and J. Wolff. von Gudenberg, "Pattern-based design recovery of Java software." In *Proceedings of 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 10-16. ACM Press, 1998.

[20] P. Tonella, G. Antoniol, "Object oriented design pattern inference." In *Proceedings of International Conference on Software Maintenance (ICSM'99)*, 1999.

[21] Blewitt and A. Bundy, "Automatic verification of Java design patterns." Proceedings of International Conference on Automated Software Engineering, pp. 324-327, 2001.

[22] H. Mei, T. Xie, and F. Yang, "JBOORET: an automated tool to recover OO design and source models." In *Proceedings of the 25h Annual International Computer Software & Applications Conference (COMPSAC)*, 2001.

[23] H. Albin-Amiot, P. Cointe, Y. Guéhéneuc, and N. Jussien, "Instantiating and detecting design patterns: putting bits and pieces together." In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE)*, 2001.

[24] Asencio, S. Cardman, D. Harris, and E. Laderman, "Relating expectations to automatically recovered design patterns." *Proceedings of the 9th Working Conference on Reverse Engineering (WCRE)*, 2002.

[25] L. Wendehals, Improving design pattern instance recognition by dynamic analysis. *Proceedings of the ICSE workshop on Dynamic Analysis*, pp. 29-32, 2003.

[26] J. M. Smith and D. Stotts, "SPQR: Flexible automated design pattern extraction from source code." In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE)*, 2003.

[27] D. Heuzeroth, T. Holl, G. Hogstrom, and W. Lowe, "Automatic design pattern detection." In *Proceedings of the 11th International Workshop on Program Comprehension (IWPC 2003)*, pp 94-103, 2003.

[28] D. Heuzeroth, S. Mandel, and W. Lowe, "Generating design pattern detectors from pattern specifications." *Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE)*, 2003.

[29] D. Beyer, A. Noack, and C. Lewerentz, "Simple and efficient relational querying of software structures." In P*roceedings of the 10th Working Conference on Reverse Engineering (WCRE'03)*, 2003.

[30] C. Park, Y. Kang, C. Wu, and K. Yi, "A static reference analysis to understand design pattern behavior." In *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04)*, 2004

[31] Z. Zhang, Q. Li, and K. Ben, "A new method for design pattern mining." In *Proceedings of the 3rd International Conference on Machine Learning and Cybernetics*, 2004.

[32] G. Costagliola, A. De Lucia, V. Deufemia, C. Gravino, and M. Risi, Design Pattern Recovery by Visual Language Parsing, Proceeding of the Ninth European Conference on Software Maintenance and Reengineering. (CSMR'05), pp. 102-111, 2005.

[33] G. Costagliola, A. Lucia, V. Deufemia, C. Gravino, and M. Risi, "Case studies of visual language based design patterns recovery." In *Proceedings of the Conference on Software Maintenance and Reengineering (CSMR)*, 2006.

[34] D. Streitferdt, C. Heller, I. Philippow, "Searching design patterns in source code." In *Proceedings of the 29th Annual International Computer Software and Application Conference. (COMPSAC)*, 2005.

[35] Heyuan Huang, Shensheng Zhang, Jian Cao, Yonghong Duan, A practical pattern recovery approach based on both structural and behavioral analysis, Journal of Systems and Software, 75(1-2), pp.69-87, February 2005

[36] W. Wang and V. Tzerpos, "Design pattern detection in Eiffel systems." In *Proceedings of 12th Working Conference on Reverse Engineering (WCRE'05)*, 2005.

[37] Kaczor, Y. Guéhéneuc, and S. Hamel, Efficient Identification of Design Patterns with Bit-vector Algorithm, Proceedings of the Conference on Software Maintenance and Reengineering (CSMR), 2006.

[38] N. Shi and R. A. Olsson, "Reverse engineering of design patterns from java source code." *21st IEEE/ACM International Conference on Automated Software Engineering*, 2006.

[39] J. Dong, D. S. Lad and Y. Zhao, "DP-Miner: Design Pattern Discovery Using Matrix." The *Proceedings of the Fourteenth Annual IEEE International Conference on Engineering of Computer Based Systems (ECBS)*, Arizona, USA, March 2007.

[40] J. Dong and Y. Zhao, Experiments on Design Pattern Discovery, *the Proceedings of the 3rd International. Workshop on Predictor Models in Software Engineering (PROMISE), in conjunction with ICSE*, USA, May 2007.

[41] J. Niere, W. Schafer, J. P. Wadsack, L. Wendehals, and J. Welsh, "Towards pattern-based design recovery." In *Proceedings of the 24th International Conference on Software Engineering (ICSE)*, pp 338-348, 2002.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 11, November-2019

[42] Y. Guéhéneuc, H. Sahraoui, and F. Zaidi, "Fingerprinting design patterns." *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE)*, 2004.

[43] R. Ferenc, A. Beszedes, L. Fulop, and J. Lele, "Design pattern mining enhanced by machine learning." In *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*, 2005.

[44] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. Halkidis, "Design Pattern Detection Using Similarity Scoring." *IEEE transaction on software engineering*, Vol. 32,No. 11, November 2006.

[45] Z. Balanyi and R. Ferenc, "Mining design patterns from C++ source code." *Proceedings of the 19th IEEE International Conference on Software Maintenance (ICSM)*, pp. 305-314, September, 2003.

[46] M. Hericko and S. Beloglavec, "A composite design-pattern identification technique." Informatica 29, pp 469–476, 2005

[47] J. Niere, J. P. Wadsack, L. Wendehals, "Handling large search space in pattern-based reverse engineering." *Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC)*, pp. 274-279, 2003.

[48] Jing Dong, Yongtao Sun, Yajing Zhao, Compound Record Clustering Algorithm for Design Pattern Detection by Decision Tree Learning, *the Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI), USA, July 2008.*