Design Pattern Detection Using Dpdetect Algorithm

Jyoti Singh, Manjari Gupta

Abstract: **Design Patterns help to solve several recurring design issues in object oriented software** . ***Expert software designers give design in terms of already proven design patterns to make their design more standard and less error-prone. Having the knowledge of design patterns used in the design helps to get insight into design. Thus*** **detection of Design Patterns is very important for software designer to get significant** ***information*** **during** ***re-engineering*** **process.** ***Detecting*** **Design Patterns from source code** ***or design of software system will*** **help in the** ***software*** **understanding and** ***its*** **maintenance.** ***It is also useful for novice developers who can get the idea about how to give solution (design) of any particular application using design patterns proposed by expert designers. Many design patterns detection approaches have been proposed by different researchers working in this field for more than two decades. These approaches consider structural, behavioural and/or semantic analysis of software system. Many sub graph isomorphism techniques were used to detect design patterns in case of structural analysis.*** **In this paper we** ***are using*** **a** ***branch and bound with backtracking*** **algorithm** ***for sub graph isomorphism, proposed by Asiler and Yazici [19]. We use this algorithm to show how this recover all the instances of design patterns from system design(renamed as DPDetect). Our main aim is to detect whether a particular design instance of design pattern is found in system design or not. It uses structural aspects of design patterns so it is based on only static analysis.***

Index Terms: Design pattern s, sub graph isomorphism, detection, structural analysis, UML.

I. INTRODUCTION

Software systems are dynamic products in the sense of future changes needed in that because of continuous changes required in most of the requirements of the system. This evolution process of software systems need changes in design elements of the system. Finding which design elements are required to be changed is always a challenge for software maintainers. Now a days software systems are developed using design patterns (already tested solution of a recurring design problem) to make their design more flexible, elegant and robust. Even in all phases of software development possibility of software reuse (design patterns, frameworks and components) is being explored [27]. Sometimes changes in requirements need changes in design elements of used design patterns (DP) also. Upgrading such software systems involves the process to understanding it that needs understanding all design patterns used in it, remaining design elements and their communications. This process involves as a first step detecting design patterns in the software system to be changed. Thus we can say detecting design patterns helps in understanding and thus maintaining software systems. A tool is therefore needed that can automatically detect design patterns used in a software system. When the knowledge about design patterns is recovered from this tool, the software developers have advantage to upgrade the design of design patterns as well as system designs.

Many graph matching algorithms have been widely used for design pattern detection techniques. In this paper we are using a sub graph isomorphism algorithm proposed by Asiler et al. [19] for DP detection. This algorithm was proposed for efficiently querying big graph databases. It is branch and bound with backtracking algorithm. Here we propose how this algorithm can be used for design pattern detection and named as DPDetect. DPDetect algorithm detects all instances of a design pattern from the source code. This approach takes UML class diagram of design patterns to be detected as well as UML design of software system from which these patterns are to be detected as inputs and outputs are instances of these design patterns, if present, in the system. This approach uses UML design of the system software and thus does only static analysis to detect design patterns.

The rest of this paper is organized as follows . Section 2 introduces similar works on design pattern detection. Section 3 describe the necessary background and motivations. Section 4 presents the DPDetect algorithm. Section 5 illustrates the algorithm using an example. Finally section 6 presents conclusion.

II. RELATED WORK

In this section, we present a comprehensive overview of the different approaches of design pattern detection based on different detection strategy to analyse the structural, behavioural and semantic aspects of source code of design patterns. Different mining techniques use different approaches to detect design pattern instances from source code.

Brown[8] was the first researcher who make an effort towards the automatic detection of design patterns. In which Small-talk code was reverse engineered to encourage the recovery of four well known patterns by Gamma et al. [1]. Some approaches [17] [18] of design pattern detection give the information about formal specification like which role is significant for composition and formalisation of design patterns. Rasool et al. [14] proposed database queries to recover the instances of design patterns.

In the last decade, graph theory have a large variety of applications to solve many practical and real life applications. It has also been

Revised Manuscript Received on May 10 ,2019

Jyoti Singh, DST - CIMS, Institute of Science, Banaras Hindu University, Varanasi, India
Manjari Gupta, DST - CIMS, Institute of Science, Banaras Hindu University, Varanasi, India

used for mining of design patterns. Vincent D. Blondel [4] have been proposed a methodology to calculate the similarity of two vertices. Nikolas Tsantalis et al. [2] introduced similarity scoring algorithm for detection of design patterns. However the main drawback of this algorithm is that it can not calculate the similarity between two graphs. It can calculates only similarity between two nodes of graph. Jing Dong [24] proposed another method Template Matching which overcomes the above technique's limitations. It calculates the similarity between subgraphs. Another method of design pattern detection is Difference calculation method [12] developed by S. Wenzel, which works on UML models. The advantages of S. Wenzel approach over other detection methodology is that it can recover incomplete pattern instances of design patterns. Bergenti and Poggi [7] proposed a methodology of design pattern detection which examines UML diagrams and developed software architect modification to the design.

Fuzzy graph approach for design pattern detection and Klenberg approach used earlier for detection of design patterns. .It has a limitation that they are not focused about whole graph and only focused on node similarity. To solve the above limitation many sub graph matching algorithm is proposed for detection. A new technique DNIT (Depth-Node-Input Table) [9] is developed for design pattern detection, it extracts design patterns which present at different level of rooted directed graph. In another work state space representation [11] of graph matching algorithm is used to detect design patterns. The advantage of this approach over other design pattern detection approaches was that the memory requirement was quite lower and it detects presence of each design pattern. Another approach [25] using greedy algorithm for multi-labeled graph is proposed for mining of design patterns.

The number of approaches [13] [6] focused on metrics such as association, inheritance, dependency, aggregation, realization, etc. of source code. The metric approach of design pattern detection is used to minimize the search space efficiently. Uchiyama et al. [20] proposed a technique which uses source code metric and machine learning for recovery of design patterns. Design pattern detection tools are used for recovery of pattern instances from source code. Many tools are used for the recovery of design patterns. J. Dong et al. [5] has been developed a tool for recovery of design pattern instances which analyses the structural, behavioural and semantic characteristics of design pattern. PINOT [21] is a fully automatic design pattern recovery tool for detection of design patterns, and focused on structural and behavioural characteristics of design patterns.

III. BACKGROUND AND MOTIVATIONS

Graph theory has been widely used to solve many problems of computer science and in particular software engineering. Maintenance is one of the most costly phases of software development. Design patterns detection may be very helpful for maintenance of software systems. Graph theory has been already applied for design patterns detection also. In our approach , we describe how design patterns and system design is represented in terms of graphs and definitions in the following sub sections.

A. Representation of system design and design patterns

UML design of software system and UML class diagram of design patterns can be converted into their corresponding directed graphs by taking nodes corresponding to classes and edges for relationships among classes. Let GDP: (U , E) represents the DP graph with the vertex set U and edge set E and GSD: (U' , E') represents the graph with vertex set U' and edge set E' of system design (SD). Nodes can be labelled by the metric values calculated for the corresponding class, for example no of variables, no of methods, fan-in and fan-out values etc. Similarly edges can be labelled by the relationship names like aggregation, association, dependency and generalization etc. We use label 'ass' for association, 'gen' for generalization, 'dep' for dependency and 'agg' for aggregation. Similarly we can create graph for any given UML design of software system. We are considering only edge's labels. Here we are not specifying the class label. Developers who will develop a particular tool using this approach will decide which class metrics they will consider and thus decide the class label.

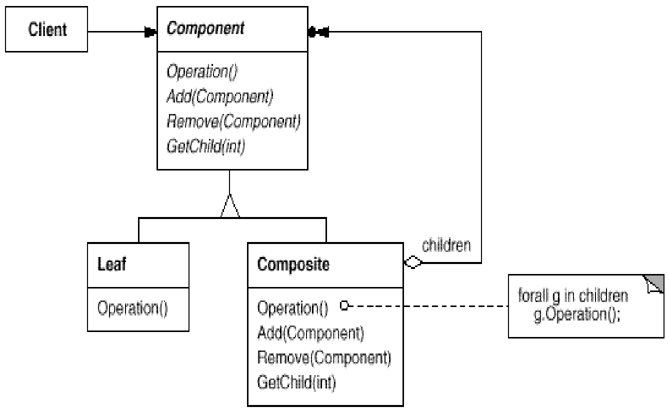

Fig.1: UML diagram of composite design pattern [1]

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

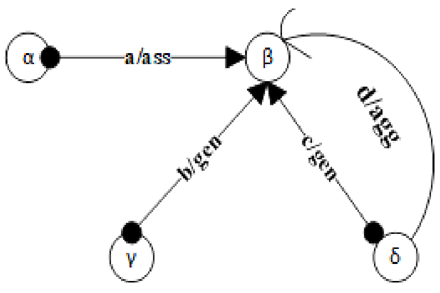

Fig.2: Graph of composite design pattern

B. Definitions used in algorithm

The algorithm used here for DP detection uses the concept of node and edge equivalence. Before defining node equivalence we will define equivalence of edges. Two edges adjacent to u and u' are equivalent if their labels as well as their direction with respect to u and u' are same. If two edges are equivalent it does not mean these two edges are mapped also. They may or may not be mapped. The algorithm will determine whether these are mapped or not. Any node (u) of GDP is equivalent to a particular node (u') of GSD if

i) Their labels are same (i.e. metric values for the classes corresponding to these nodes are same)

ii) Degree (each group of edges having same label and direction) of u is less than degree of u'.

iii) Each edge adjacent to u is also equivalent to corresponding edge adjacent to u'.

Algorithm maps one by one nodes and edges of DP to SD and finds instances of design patterns. In this process few nodes and edges will be mapped but if full design pattern instance cannot be found then these mappings will be cancelled. If whole instance is found, these mappings are treated as final mappings.

IV. DPDetect algorithm

The algorithm first chooses one node (u) of DP graph randomly. It then finds all nodes (u') of SD graphs that are equivalent to this first chosen node of DP graph. It then starts finding whole instance of design pattern in the system design by taking these two nodes (u and for each u') as root nodes and extending it to whole DP graph and corresponding graph in SD graph. For this node u, it considers one by one each edge (e) that is non mapped and adjacent to u. For each e it finds all edges (e') adjacent to u' that are equivalent to e.

For each e and e' that are equivalent it will consider the other end nodes (v and v'). Now there are three cases.

i) If only one of these other end nodes are already mapped to any other node of the other graph, e can not be mapped to e'. Thus the other e' that is equivalent to e will be considered if any. If no other e' exists that is equivalent to e, whole instance can not be formed rooted with u'. The whole process will start for next u' if it exists.

ii) If both end nodes are already mapped to each other, e is mapped to e' and it will consider the other edge e adjacent to u and not mapped till now.

iii) if none of these nodes v and v' are mapped, after processing all edges adjacent to u the algorithm will be applied on this pair v and v' and the same process continues until no new v and v' are found. This process is explained through algorithm that is shown below.

Algorithm DPDetect

1.Choose one of the node from DP graph: N1

2.Find all possible nodes (Mi) of system design equivalent to N1

3.For each node Mi equivalent to N1, call FindInstance(N1, Mi) to find one of the instance of Design pattern represented as DP graph in SD graph .

Algorithm: FindInstance(Ni, Mi)

1. Find all non mapped relationships r_i adjacent to N_i
2. For each r_i, find all relationships r_i' adjacent to M_i and equivalent to r_i
3. For each relationship r_i, call check (r_i, r_i') to find if r_i can be mapped to r_i'. If not, do the same for next r_i'.
4.If all r_i are mapped to any corresponding r_i', map N_i to M_i.

Algorithm: Check()

1. Determine the other end nodes (v_i and v_i') of r_i and r_i'
2. If other end nodes v_i and v_i' of r_i and r_i' respectively are already in M as a pair, r_i is mapped to r_i'.
3. Else If one of them is in one of the pair in M and other one is not present in M, r_i cannot be mapped to r_i'.
4. Else if none of these are in M, it is checked whether v_i and v_i' are equivalent. If yes, v_i is mapped to v_i' and r_i is mapped to r_i' and call FindInstance(v_i, v_i') when all relationships adjacent to N_i will be covered.

V. Illustration of algorithm on a simple example

In this section the algorithm is explained using one simple example. We are showing how different instances of composite design pattern will be searched in the system design, if these are present. UML diagram as well as the corresponding DP graph of composite design pattern are shown in fig. 1 and fig. 2 respectively. UML diagram and its SD graph of an arbitrary system design taken as a sample in this paper are shown in fig. 3 and fig. 4 respectively. Fig. 5 explains how detection algorithm works for this example.

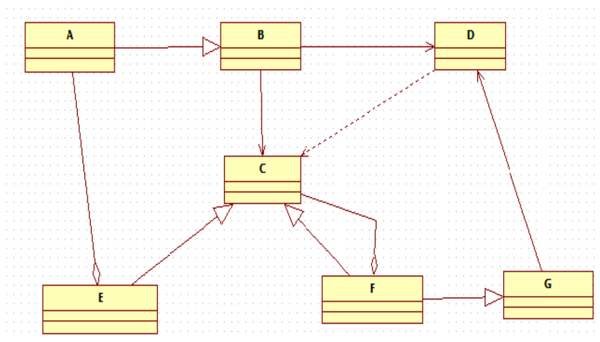

Fig. 3: UML diagram of System Design[23]

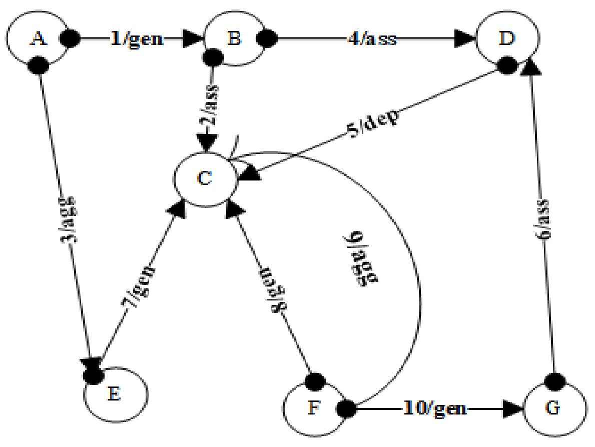

Fig. 4: Graph of System Design

Detection process starts with algorithm DPDetect that randomly chooses a node, let α, as the first node from DP graph shown in Figure 2. Node B and G of SD graph shown in figure 4 are equivalent to node α. Then this algorithm will call two times FindInstance for (α , B) and (α , G) to find two possible instances of composite design patterns in systems design. Tracing of both of these calls to FindInstance is explained below one by one.

i) FindInstance(α , B) searches all edges adjacent to α. There is only one edge a adjacent to α and not mapped. Edges 2 and 4 are adjacent to B that are non mapped as well as equivalent to edge a. It will call i.1) Check(a,2) and i.2) Check(a,4).

i.1) Check(a, 2) will determine the other end nodes of edges a and 2. These are nodes β and C. Both of these nodes β and C are not already mapped to any other nodes and these are equivalent. So β is mapped to C and the algorithm FindInstance will be called for β and C.

FindInstance(β,C) finds three edges b, c and d that are non mapped and adjacent to β. Edges that are adjacent to node C and non mapped as well as equivalent to edge b are edges 7 and 8. Possibility of mapping of b to both 7 and 8 will be checked.

i.1.1 First it will consider edges b and 7. There other end nodes are γ and E. Both are not mapped to any other node and these are equivalent. So γ is mapped to E. FindInstance will be called for (γ, E).

Before FindInstance(γ, E) is called the next non mapped edge, let c, that is adjacent to node β is taken. It is equivalent to edge 8 that is non mapped and adjacent to node C. δ and F are other end nodes of edges c and 8. Both δ and F are not mapped to any other node and are equivalent so δ is mapped to F. FindInstance will be called for δ and F.

Before FindInstance(δ ,F) is called the next edge adjacent to β that is non mapped is d. It is equivalent to edge 9 and 5 adjacent to C that are not mapped.

i.1.1.1 To consider edges d and 9, the other end nodes of d and 9 are δ and F. Both are already mapped to each other. So edge d is mapped to edge 9. Now no edge adjacent to β remains. Here we got one instance of composite design pattern in sample system design.

i.1.1.2 Now it will consider edge d and 5. Other end nodes of these are δ and D. Both of these nodes are not equivalent so edge d cannot be mapped to edge 5.

i.1.2 Now edges b and 8 will be considered for mapping. Other end nodes of these are γ and F. These are equivalent. So edge b is mapped to edge 8 and node γ is mapped to node F. FindInstance will be called for γ and F. Before that next edge that is non mapped and adjacent to node β will be taken-let c. c is equivalent to 7. Other end nodes are δ and E. Both are not equivalent thus this node can not be further extended as shown in figure 5.

i.2) Check(a, 4) will determine the other end nodes of edges a and 4. These nodes are β and D. Both of these nodes are also not mapped to any other node. Both of these are not equivalent so β cannot be mapped to D and thus a can not be mapped to 4.

ii) FindInstance(α,G) finds edge a that is non mapped and adjacent to α. Edge 6 is non mapped, adjacent to G as well as equivalent to a, thus it will call Check(a,6). Other end nodes of edges a and 6 are β and D. These are not equivalent and thus node β cannot be mapped to G and edge a cannot be mapped to 6. Thus this node cannot be extended further.

VI. CONCLUSION AND FUTURE WORK

This paper presents a branch and bound with back tracking algorithm (DPDetect) to detect design patterns from software system. Originally this algorithm for subgraph isomorphism was proposed by Asiler and Yazici [19] foe efficiently querying big graph databases. It is very difficult to find all exact matches of design patterns in a system design. This approach searches all the instances of design patterns in system graph. We shown how this algorithm detects instances from software system by a simple example of composite design pattern.

The input is graph corresponding to UML diagrams of design patterns as well as system design and output is all occurrences of design patterns in the system design. If there exist any instance of design patterns in system graph the algorithm match one by one each node and its adjacent edges of DP to the nodes and adjacent edges of these in SD. The next node to be matched is the other end node of one of the edge adjacent to the current node. Criteria for selection of next node is explained above in the section IV.

The main advantages of this algorithm over few other existing algorithm is that it will extract every instances of design patterns. However, the main drawback of our approach is that it can only analyse structural aspect of design patterns. We are extending this approach to incorporate behavioural as well as semantic aspects of design patterns. Further node labels need to be added to get better results. We are developing a tool parallely that implements the algorithm described in this paper so that performance of this algorithm can be evaluated over other algorithms.

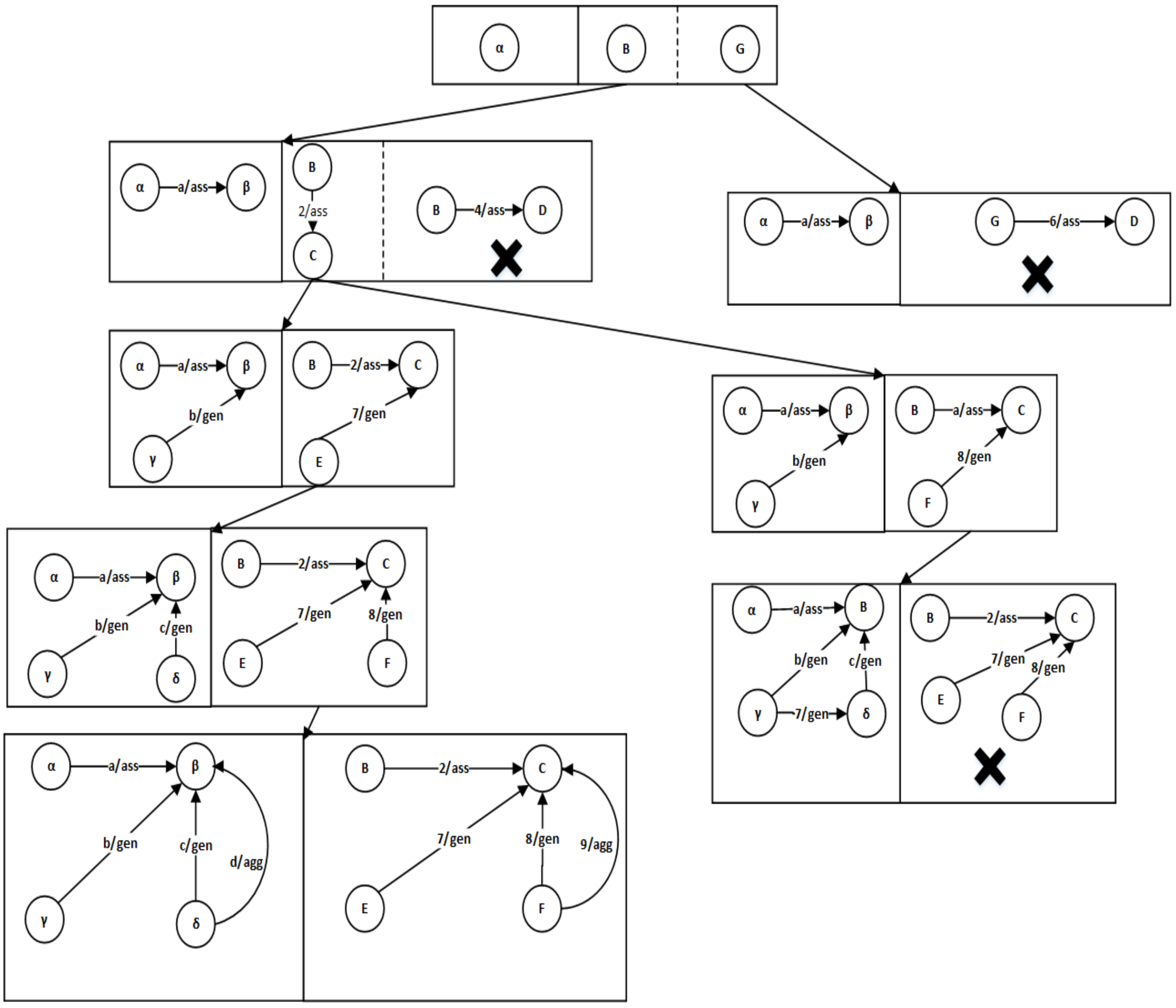

Fig. 5: Illustration of algorithm for detection of composite design pattern in system design

REFERENCES

1. E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns Elements of Reusable Object-Oriented Software.Addison-Wesley, Reading(1995).
2. N. Tsantalis, A.Chatzigeorgiou, G. Stephanides, and S. Halkidis, "Design Pattern Detection Using Similarity Scoring," IEEE transaction on software engineering, 32(11), 2006.
3. G. Antoniol, G. Casazza, M. Di Penta, and R. Fiutem, " Object-Oriented Design Patterns Recovery", J. System and Software,vol. 59,no.2, pp. 181-196,2001.
4. 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.
5. 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), USA, March 2007.
6. H. Bunke and B. T. Messmer. " Recent advances in graph matching. Int. J. Pattern Recognition and Artificial Intelligence, 11(1):169{203, 1997.
7. F. Bergenti and A. Poggi, Improving UML Designs Using Automatic Design Pattern Detection, Proc. 12th Int'l Conf. Software Eng. and Knowledge Eng.(SEKE'00), July 2000.
8. K. Brown, " Design Reverse-Engineering and Automated Design Pattern in Smalltalk", Technical Report TR-96-07,Dept. Of Computer Science,North Carolina State Univ.,1996.
9. A. Pande, M.. Gupta, A.K. Tripathi "DNIT – A New Approach for Design Pattern Detection", International Conference on Computer and Communication Technology, MNNIT- Allahabad, proceeding published by the IEEE, 2010.
10. A. Pande & M. Gupta, "Design Pattern Mining for GIS Application using Graph Matching Techniques", 3rd IEEE International Conference on Computer Science and Information Technology. pp. 09-11, Chengdu, China, 2010.
11. M. Gupta, R.R. Singh, A. Pande, A.K.Tripathi,"Design pattern Mining Using State Space Representation of Graph Matching", 1st International Conference on Computer Science and Information Technology, Banglore, 2011, to be published by LNCS, Springer.
12. S. Wenjel, U. Kelter, "Model-driven design pattern Detection using difference calculation method", In Proc. of the 1st International Workshop on Pattern Detection for Reverse Engineering(DPD4RE), Benevento, Italy,2006.
13. M. V. Detten, and S. Becker, "Combining Clustering and Pattern Detection for the Reengineering of Component-based Software Systems", In Proceedings of the 7th International Conference on the Quality of Software Architectures, QoSA, pp. 23-32, 2011.
14. G. Rasool, I. Philippow, P. Mader, "Design Pattern Recovery Based on Annotations". International Journal of advances in Engineering Software, Vol 41, Issue 4, 2010, pp. 519- 526.
15. J. Dong, J. Zhao, and Y. Sun, " A Matrix based Approach to Recovering Design Patterns", IEEE transactions on Systems, Man and Cybernatics, Vol 39, No. 6, 2009, pp. 1271-1282.

16. J. Paakki, A. Karhinen, J. Gustafsson, L. Nenonen, and A. I. Verkamo, "Software Metrics by Architectural Pattern Mining", In Proceedings of the International Conference on Software: Theory and Practice, 2000, pp. 325–332.
17. Y. Wang, and J. Huang, "Formal Modeling and Specification of Design Patterns Using RtPA", International Journal of Cognitive Informatics and Natural Intelligence, Volume 2, Issue 1, 2008, pp. 100-111 .
18. Alnusair, A., Zhao, T., Yan, G., 2014. "Rule based detection of design patterns in program code.". Int.J.Softw.ToolsTechnol.Trans.16 (3), 315–334.
19. Merve Aslier and Adnan Yazici, Senior Member, IEEE, " BB-Graph: A Subgraph Isomorphism Algorithm for Eficiently Quering Big Graph Databases.
20. Uchiyama, S., Kubo, A., Washizaki, H., and Fukazawa, Y. (2014). Detecting Design Patterns in Object-Oriented Program Source Code by Using Metrics and Machine Learning. Journal of Software Engineering and Applications, 7, 983-998. doi: 10.4236/jsea.2014.712086.
21. Nija Shi and Ronald A. Olsson. Reverse engineering of design patterns from java source code. In ASE '06: Proceedings of the 21st IEEE International Conference on Automated Software Engineering (ASE'06), pages 123–134, Washington, DC, USA, 2006. IEEE Computer Society.
22. Pande A., Gupta M., "Design Pattern Detection Using Graph Matching", International Journal of Computer Engineering and Information Technology (IJCEIT), Vol15, No 20, Special Edition, pp. 59-64, 2010.
23. StarUML, The Open Source UML/MDA Platform. http://staruml.sourceforge.net/en/.
24. 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.
25. Manjari Gupta (2011), Design Pattern MiningUsing Greedy Algorithm for Multilabeled Graphs, International Joint Conference on Information and Communication Technology, 8- Jan, 2011, Bhubaneshwar, to be published by IPM Pvt. Ltd, India.
26. M. Gupta, A.Pande, "Design Patterns Mining using Subgraph Isomorphism: Relational View", International Journal of Software Engineering and Its Applications Vol. 5 No. 2, April, 2011.
27. M. Gupta, M. Prakash, "Possibility of Reuse in Software Testing", 6th annual International Software Testing Conference in India 2006.

AUTHORS PROFILE

Miss Jyoti Singh pursuing her PhD in DST - Centre for Interdisciplinary Mathematical Sciences, Institute of Science, Banaras Hindu University, Varanasi and has her MSc degree in Computer Science from Department of Computer Science, Institute of Science, Banaras Hindu University, Varanasi, India. She is engaged in research for the last 3 years and the areas of her interest is Design Pattern Detection.

Dr. Manjari Gupta is serving as Associate Professor with the DST - Centre for Mathematical Sciences, Institute of Science, Banaras Hindu University,Varanasi, India. She has her MSc degree in Computer Science from J. K. Institute of Applied Physics and Technology, Allahabad University, Allahabad and the PhD in Computer Engineering from Indian Institute of Technology, Banaras Hindu University, Varanasi, India. She is engaged in teaching and research from the last 15 years and the areas of her research interests include software reuse, object oriented frameworks and design pattern detection.

.