Volume 64, Issue 2, 2020

Journal of Scientific Research

Institute of Science,

Banaras Hindu University, Varanasi, India.

Mining Design Patterns using String Encoding Format

Jyoti Singh[1], Sripriya Roy Chowdhuri[1], Bethany Gosala[1], Akshara Pande[2], and Manjari Gupta[*1]

[1]DST- CIMS, Institute of Science, Banaras Hindu University, Varanasi, India.
jyotisingh4337@gmail.com, sripriyaroychowdhuri@gmail.com, bethany.beta777@gmail.com, manjari@bhu.ac.in*
[2]School of Computing, Graphic Era Hill University, Dehradun, India. pandeakshara@gmail.com

Abstract: **In object oriented software, design pattern gives the particular solution for common design problems. In software engineering it's very difficult task to find out design information due to improper documentation of software systems. It is very much necessary to recover pattern instances so that system would be understandable and can do modifications in them. Actually recovery of design patterns play a significant role in object oriented programming for software developers and researchers during development of system software and their maintenance. Hence mining of design patterns are very important. The paper describes detection of design patterns from software or system design by using the String Encoding Format in which pattern and system graphs are transformed in string after that process of matching is performed to extract instances from software systems. Here we match string of system design graph and design pattern graph using structural analysis.**

Index Terms: **UML, String Encoding Format, Relationships, Design Patterns, Reliability.**

I. Introduction

Design patterns solve common issues of frequent design problems (Gamma 1995). The expert of software systems and researchers reuse the design which play significant role in software industry and reduce the effort and time of software developers. Since requirements of software systems are always changing, there is a need of modification (maintenance) in the software.

To develop and maintain reliable software, one of the requirements is to have a complete idea of design patterns existing in the source code. Sometimes information of used design patterns can give the idea about whole software documentation. Thus with the help of design pattern mining reliability of software can be maintained during its modification. If design-patterns could be captured and reused it gives an idea to the developers and maintainers of software which is very useful. In recent years design pattern attract researchers towards mining of design patterns as design patterns encapsulate valuable knowledge and information about system design. Mining of design patterns play an important role in re-engineering process, during program and system understanding, during maintenance of software systems.

Here relationship graph of system design or software and design patterns which has directed relationship. Firstly, String Encoding Format (Zaki 2005) is written for both the directed graphs. Here we find out whether design pattern exists in system design string or not. The graph representation of UML diagram is shown in section 2. The String Encoding Format of relationship graphs is described in section 3. Design pattern detection is described in section 4. Related works are described in section 5 and finally we conclude in section 6.

II. GRAPH REPRESENTATION

For any software system its UML diagram, particularly class diagram is available with this design document. In this technique we use the UML diagram to represent both design patterns (DP) as well as system design (SD). After that these diagrams are converted into corresponding graphs. The UML diagram of SD and DP is taken first and extract the relationship directed graphs corresponding to both SD and DP to match the string. Here we are taking SD and their corresponding relationship graphs which have been shown in figure 1, 2, 3 and 4.

[*] Corresponding Author

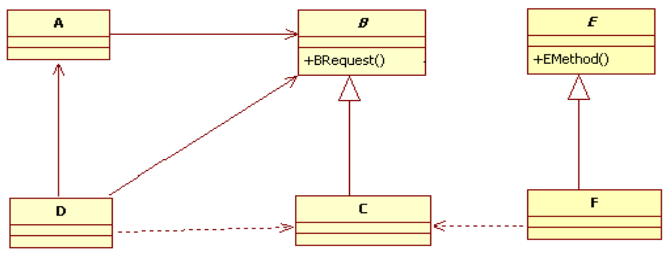

Figure 1. UML Diagram: System design

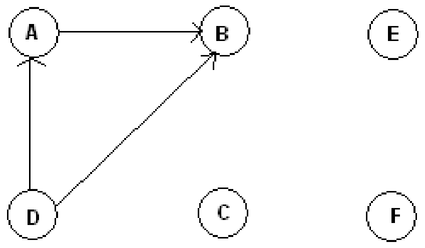

Figure 2. Direct association relationship graph: System design.

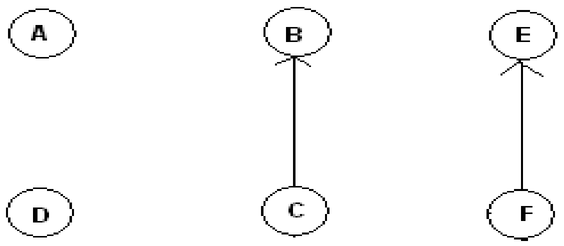

Figure 3. Generalization relationship graph: System design.

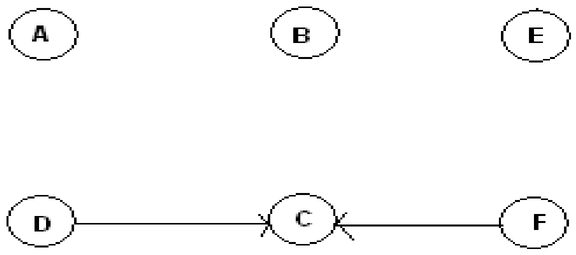

Figure 4. Dependency relationship graph: System design.

In similar way, relationship graphs for DP can be extracted. We are considering two design patterns, i.e. façade design pattern and factory method design pattern, and their corresponding relationships, shown in figure 5 -figure 9.

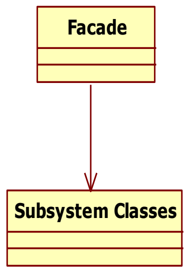

Figure 5. UML diagram: Façade design pattern.

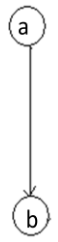

Figure 6. Direct association relationship graph: Façade design pattern.

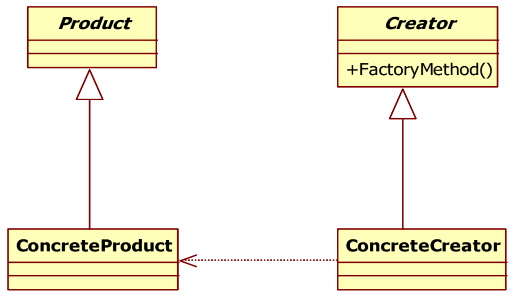

Figure 7. UML diagram: Factory method design pattern.

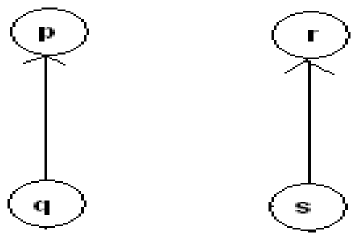

Figure 8. Generalization relationship graph: Factory method design pattern.

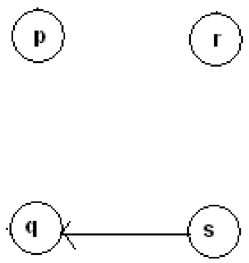

Figure 9. Dependency relationship graph: Factory method design pattern.

III. Methodology

Here we show the String encoding method for a directed graph: Let us consider the directed graph shown in figure-10. Here we extended the Regular Continuous Directed Graph (zaki 2005), Sreenivasa and Ananthanarayana(2006) with some modifications.

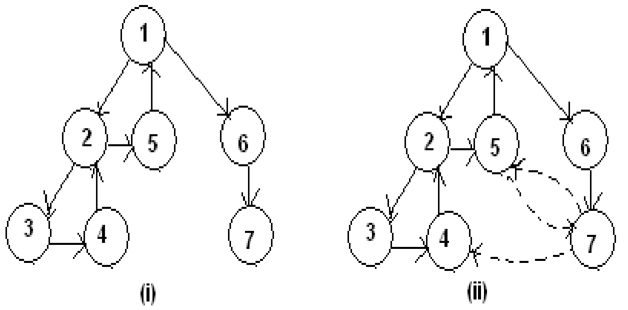

Figure 10. Directed graph.

Here in the figure-10 (i) 1 2 3 4-2 5-1 6 7 show the form of encoded string. After reaching on node-7 there is no node to visit again. In figure-10 (ii) three dummy edges are introduces (ie. 7->4, 7->5, 5->7). Now suppose we want to write String Encoding Format for this, then the String Encoding Format would be: 1 2 3 4-2 5-1 6 7 0-5 0-7 0-4. Here '0' is used to show the dummy edge and '-' is used to show that the is revisited again. Now the same strategy is used to write the String Encoding Format for the relationship directed graphs of SD and DP.

IV. STRING ENCODING FORMAT For Design Patterns

Here first we consider about Façade design-pattern (represented by figure 5), where one relationship is found which is direct association (figure 6). String Encoding Format for this is ***a b.*** Similarly for factory method design pattern, the String Encoding Format for dependency relationship is ***s q***. But for generalization there are two occurrences of the same relationship. Now reachable path has to be found which covers the entire node at least once by introducing dummy edges as shown in figure 11. Then the corresponding String Encoding Format is ***q p 0 r 0 s 0 -q 0 –s –r***.

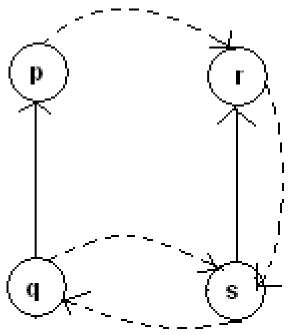

Figure 11. Generalization relationship graph of factory method design pattern including dummy edges.

V. STRING ENCODING FORMAT For System Design

SD has three relationship graphs, as shown in figure 2, 3 and 4. For direct association relationship the String Encoding Formats are ***A B, D A, D B*** (here we have not introducing any dummy vertices because only two of design patterns have been considered, in which only one design pattern i.e., Façade design-pattern has String Encoding Format of length two). For generalization relationship the String Encoding Format is ***C B 0 E 0 F 0–C 0–F–E*** (the procedure is the same as for figure 11). For dependency relationship, String Encoding Format is ***D C*** and ***F C***.

VI. DETECTION Algorithm

In this section detection algorithm is described in detail.

Matching Algorithm:

```
Input: UML diagrams of SD and DP.
Output: 1 if design pattern exists for a relationship
Extract Relationship graphs ($G_i$s) form UML of SD and DP. Introduced dummy edge in $G_i$s if there are two edges $e_{ij}$ and $e_{kl}$ where i, j, k, l are all distinct nodes.
Write string encoding format for the $G_i$s exist in design patterns.
For each relationships exist in design-pattern
  Find out the string length (say n).
  Write string encoding format (for length n) for the same relationship in system design.
  Compare the strings of design pattern and system design.
If strings are equal then design pattern exists return 1
End for
End Algorithm Matching
```

Now consider the String Encoding Format corresponding to DP and SD. Suppose we want to find out Facade design-pattern in system design, the String Encoding Format for Façade design-pattern is ***a b*** which is the same as of system design i.e., ***A B, D A, D B***. This shows the occurrence of Façade design-pattern in system design. Similarly, for factory method design pattern for each relationship of design pattern the String Encoding Format of system design is the same. Hence instances of factory method exist in system design. Here Façade and factory method design patterns exists completely in system design. By applying above discussed algorithm we found 1 for each relationship, so there is complete match. There are three cases found during detection of patterns in system design in first case all the relationships of patterns are matched with SD which is mentioned earlier known as complete match. Another possibility is that only few relationships of patterns are found in system design in that case we found output 1 for some of the relationships; it is called partial detection of design pattern in the last case there are chances that no relationship of patterns are matched with any relationships of SD, means no match. In this case we never found output 1 for any of the relationships.

VII. RELATED WORK

Mining of design patterns are very popular among the researchers and software developers. Mining of design patterns are based on static and dynamic analysis of design patterns. Structural or behavioural properties of design patterns. Structural approaches recognize the structural aspects of patterns such as class names their relationships such as association generalization, realization, dependency, methods and attributes of classes. Behavioural techniques are based on procedure of programs such as which class calls other classes.

They play significant role in comparing the patterns which are identical. But sometimes structural and behavioural approaches are unable to detect those patterns which are structurally identical like state and strategy. Semantic analysis approaches are used to detect for this type of design patterns. There are some technique which are used to distinguished similar structure design patterns.

Brown (1996) first gives the idea of automatic detection of design patterns where Small-talk code was reverse-engineered for mining of design patterns. Several researchers classified design patterns into different mining techniques to detect design patterns such as graph based approach, quantitative approach, metric based approach, machine learning based approach, constraint satisfaction approaches, formal approaches etc.

Quantitative approaches techniques are based on metrics which calculate different type of metrics such as generalization, aggregations, association, etc. and use various methodology to compare and match the metric value. These techniques are very efficient because of filtration phase. The limitation of this technique is that behavioural characteristics of design patterns are not considered and have another limitations such as low precision, low recall and lack of interactivity. A metric based approach developed by Issaoui et al. (2015) semantic and structural analysis. Metric based approach are efficient as it reduces search space.

Tsantalis et al. (2006), proposed a detection approach similarity scoring in which graph are used to represent the structure of patterns and software systems. The detection process include calculation of similarity score of matrix of design pattern and software. The main drawback of this algorithm is that it only calculates the similarity of vertices, not the similarity of graphs. Dong et al. (2008) gave an approach template matching, which solve the limitation of similarity scoring approach. A technique given by Antoniol et al. (2001) recognize structural patterns and specify the usefulness of mining tool to understand the software system.

Wenzel and Kelter (2006) proposed very popular approach called difference calculation method, the advantage of this methodolgy over other technique is that it can also find out those instances of patterns which are not complete.

Several techniques are used in our earlier work Pande et al. (2010a, b, c, d, e)), to find the instances of patterns. Singh and Gupta (2019) introduced another tool for mining of design patterns using subgraph matching with branch and bound techniques.

Some researchers detect number of design pattern by using database queries to extract information of design patterns. In query based approach SQL queries are used to extract information of design paterns using intermediate data representation. Rasool et al. (2010) presented a query based approach to extract instances of design patterns which is based on regular expression and annotation.

Some researchers also work on constraint programming for mining in which the problem of pattern detection is translated to solve the problem after that design instances are described as constraint system.

Gueheneuc et al. (2010) proposed an approach that is based on constraint programming and machine learning, the main objective of their research is to use machine learning technique to improve the performance of detection tool and to reduce search space.These approaches ensure high recall.

Guéhéneuc and Antoniol (2008) propose a tool DeMIMA which is based on multilayer approach for design pattern detection. This is semiautomatic approach which is based on static as well as dynamic analysis.

Mayvan and Rasoolzadegan (2017) proposed an approach that uses Prolog and First Order Predicate Logic (FOPL) languages.

Mhawish and Gupta (2019) uses software metrics and machine learning algorithm to distinguish those design patterns which has similar structure.

Yarahamdi et al. (2020) presented a systematic review on design pattern detection which covers various aspect of detection, design pattern detection approaches, tool, dataset used for evaluation, data representation and many more which are useful for software developers or researchers.

CONCLUSIONS

In this paper we have introduced a matching algorithm based on String Encoding Format to find the existence of patterns. We first convert relationship directed graphs corresponding to system design and design patterns into String Encoding Format. The length of string Encoding Format should be the same for both the system design and design patterns. After applying algorithm discussed in section 4, we can have the idea of all the design patterns occurrences. Thus this algorithm can always be used to improve the reliability of software. In future we are focusing on implementation of this algorithm so that performance of this algorithm can be compared with other existing design pattern mining tool.

REFERENCES

Albin-Amiot, H., Cointe, P., Guéhéneuc, Y. G., & Jussien, N. (2001, November). Instantiating and detecting design patterns: Putting bits and pieces together. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)* (pp. 166-173). IEEE.

Alnusair, A., Zhao, T., & Yan, G. (2014). Rule-based detection of design patterns in program code. *International Journal on Software Tools for Technology Transfer*, *16*(3), 315-334.

Antoniol, G., Casazza, G., Di Penta, M., & Fiutem, R. (2001). Object-oriented design patterns recovery. *Journal of Systems and Software*, *59*(2), 181-196.

Brown, K. (1996). *Design reverse-engineering and automated design-pattern detection in Smalltalk*. North Carolina State University. Dept. of Computer Science.

Dong, J., Sun, Y., & Zhao, Y. (2008, March). Design pattern detection by template matching. In *Proceedings of the 2008 ACM symposium on Applied computing* (pp. 765-769).

Gamma, E. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Education India.

Guéhéneuc, Y. G., Guyomarc'h, J. Y., & Sahraoui, H. (2010). Improving design-pattern identification: a new approach and an exploratory study. Software Quality Journal, 18(1), 145-174.

Guéhéneuc, Y. G., & Antoniol, G. (2008). Demima: A multilayered approach for design pattern identification. *IEEE transactions on software engineering*, *34*(5), 667-684.

Gupta, M., Rao, R. S., Pande, A., & Tripathi, A. K. (2011, January). Design pattern mining using state space representation of graph matching. In *International Conference on Computer Science and Information Technology* (pp. 318-328). Springer, Berlin, Heidelberg.

Issaoui, I., Bouassida, N., & Ben-Abdallah, H. (2015). Using metric-based filtering to improve design pattern detection approaches. *Innovations in Systems and Software Engineering*, *11*(1), 39-53.

Mayvan, B. B., Rasoolzadegan, A., & Yazdi, Z. G. (2017). The state of the art on design patterns: A systematic mapping of the literature. *Journal of Systems and Software*, *125*, 93-118.

Mayvan, B. B., & Rasoolzadegan, A. (2017). Design pattern detection based on the graph theory. *Knowledge-Based Systems*, *120*, 211-225.

Mhawish, M. Y., & Gupta, M. (2019). Generating Code-Smell Prediction Rules Using Decision Tree Algorithm and Software Metrics.

Pande, A., & Gupta, M. (2010a). Design pattern detection using graph matching. *International Journal of Computer Engineering and Information Technology (IJCEIT)*, *15*(20), 59-64.

Pande, A., Gupta, M., & Tripathi, A. K. (2010b). Design pattern mining for GIS application using graph matching techniques. In *2010 3rd International Conference on Computer Science and Information Technology* (Vol. 3, pp. 477-482). IEEE.

Pande, A., Gupta, M., & Tripathi, A. K. (2010c). A new approach for detecting design patterns by graph decomposition and graph isomorphism. In *International Conference on Contemporary Computing* (pp. 108-119). Springer, Berlin, Heidelberg.

Pande, A., Gupta, M., & Tripathi, A. K. (2010d). A decision tree approach for design patterns detection by subgraph isomorphism. In *International Conference on Advances in Information and Communication Technologies* (pp. 561-564). Springer, Berlin, Heidelberg.

Pande, A., Gupta, M., & Tripathi, A. K. (2010e). DNIT—A new approach for design pattern detection. In *2010 International Conference on Computer and Communication Technology (ICCCT)* (pp. 545-550). IEEE.

Rasool, G., Philippow, I., & Mäder, P. (2010). Design pattern recovery based on annotations. *Advances in Engineering S*oftware, 41(4), 519-526.

Rasool, G., & Mäder, P. (2011, November). Flexible design pattern detection based on feature types. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)* (pp. 243-252). IEEE.

Singh, J., & Gupta, M. (2019, May) Design Pattern Detection Using Dpdetect Algorithm. *International Journal of Innovative Technology and Exploring Engineering.*

Sreenivasa, G. J., & Ananthanarayana, V. S. (2006, December). Efficient mining of frequent rooted continuous directed subgraphs. In *2006 International Conference on Advanced Computing and Communications* (pp. 553-558). IEEE.

Tsantalis, N., Chatzigeorgiou, A., Stephanides, G., & Halkidis, S. T. (2006). Design pattern detection using similarity scoring. *IEEE transactions on software engineering*, *32*(11), 896-909.

Vokác, M. (2006). An efficient tool for recovering Design Patterns from C++ Code. *J. Object Technol.*, *5*(1), 139-157.

Wenzel, S., & Kelter, U. (2006, October). Model-driven design pattern detection using difference calculation. In *Workshop on Pattern Detection for Reverse Engineering*.

Yarahmadi, H., & Hasheminejad, S. M. H. (2020). Design pattern detection approaches: a systematic review of the literature. Artificial Intelligence Review, 1-58.

Zaki, M. J. (2005). Efficiently mining frequent trees in a forest: Algorithms and applications. *IEEE transactions on k*nowledge and data engineering, 17(8), 1021-1035.

****