2016 IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engineering (UPCON)
Indian Institute of Technology (Banaras Hindu University) Varanasi, India, Dec 9-11, 2016

Applying Software Metrics for the Mining of Design Pattern

Ashish Kumar Dwivedi[1], Anand Tirkey[2], Santanu Kumar Rath[3]
Department of Computer Science and Engineering
National Institute of Technology Rourkela,
769008, Odisha, India
Email: shil2007@gmail.com[1], andy9c@gmail.com[2], and skrath@nitrkl.ac.in[3]

Abstract—Development of desired software in the present day scenario is becoming too much complex as the user requirements becoming complex day-by-day. Hence there is a need for developing the right methodology for solving complex problem. To solve various problems in design phase, a number of tools and techniques are available and one of them is known as the use of design pattern, which helps to find a better solution for the problems, which are recurring in nature. It is often desired to detect design patterns from the source code of similar category of software, as it improves maintainability of source code of a software. In this study, mining of design pattern technique has been presented, which is based on supervised learning techniques as well as software metrics. During the pattern mining process, metrics-based dataset is developed. Subsequently, machine learning techniques such as Layer Recurrent Neural Network and Random Forest are applied for the pattern mining process. For the critical examination of the proposed study, data from an open source software e.g., JUnit is considered for the mining of software patterns.

Keywords—Design patterns, Design Pattern Mining, Software Metrics, Supervised Learning Methods.

I. INTRODUCTION

Software design pattern is a collection of artifacts which are applied to exchange knowledge across a particular domain. A good number of patterns have already been proposed for designing and developing a particular system [1] [2] [3]. Existing software patterns are analyzed by considering various tools and techniques [4] [5] [6] [7]. Software patterns become useful for simplifying the development of required software. The specification of software patterns is often performed by using pattern template elements, which describe solution for the recurring design problem. Pattern's template is used by Gamma et al. [1] for their design patterns. According to authors, software patterns vary in their granularity as well as the level of abstraction. Therefore the patterns catalog is organized into three classes such as creational pattern, structural pattern and behavioral pattern. These software patterns also help to improve software maintenance by making explicit representation of class and object interactions.

Design patterns are often used in the field of forward engineering, where various systems are developed by using pattern-based solution. But design patterns are often specified by considering a semi-formal approach i.e., UML (Unified Modeling Language), which provides various implementation templates for design patterns. These implementation templates are often ambiguous in nature, which create inconsistencies during the development of a system. Various formal modeling notations are proposed for analyzing pattern-based implementation template [5] [8]. The formalization of design patterns can be performed at the early phases of software development process. Detection of software pattern is a useful activity, which support the process of software maintenance by applying a suitable reverse engineering technique [7].

In this study, mining of software patterns is performed by considering a classification based approach. It is observed that design pattern mining is an useful activity that helps to support software maintenance by applying a suitable reverse engineering approach [7]. Identification of design pattern is similar to other data mining approach and it has a strong impact on affecting accuracy; which depends on values of false positive as well as false negative [9]. In the process of design pattern mining, actual instances of software design patterns available in open source software need to be identified, which helps to develop and maintain its documentation.

The proposed study considers supervised learning-based classification methods for the mining of software design patterns [1] such as Abstract Factory, Adapter, Bridge, Singleton, and Template Method from an open source software i.e., JUnit [10]. In order to perform mining process, classification methods such as Layer Recurrent Neural Network (LRNN) [11] and Random Forest [12] are used. A pattern-based dataset is prepared for the training of classifiers, which help to reduce the impacts of false positive as well as false negative, resulting in improvement of accuracy. The presented design pattern mining approach is based on learning process, which uses dataset based on object-oriented metrics. These metrics help to measure object-oriented software properties which are often considered desirable during the development processes.

The rest of the work is presented as follows. Section II describes about existing related techniques. Section III presents basic concept about the considered techniques such as software design pattern, object-oriented metrics, and machine learning techniques. Section IV highlights a two-fold approach, where firstly dataset is prepared and secondly design pattern mining process is carried out. In section V, experimental results are presented. Section VI shows a comparative analysis of the proposed study in the presence of existing related work. Finally the study is concluded.

II. RELATED WORK

Various design pattern mining techniques are available in the literature [9].

Tsantalis et al. [4] have presented the process of pattern identification that is based on similarity scoring applied among the vertices of a graph that represents a design pattern graph. They have applied similarity algorithm to detect design patterns which correspond to a piece of code residing in one or more inheritance hierarchies. Gueheneuc et al. [13] have presented an experimental approach for minimizing the search space for the design pattern instances. They have investigated several open source software manually for identifying pattern instances. Further they have applied machine learning algorithm to recognize classes playing certain roles. Dong et al. [14] have applied template matching technique for the recognition of design patterns from open source projects by computing their normalized cross correlation. According to them, their methodology identifies exact matches of design pattern instances as well as variation of patterns candidates.

Gupta et al. [15] have described an approach to identify software patterns by considering Normalized Cross Correlation while considering software pattern as a template to determine its presence in the design of a system. Ba-Brahem and Qureshi [16] have proposed an approach to detect design pattern instances in a system design which uses the graph implementation to produce both the system as well as the design pattern UML diagrams in Graph of 4-tuple elements. Gupta et al. [17] have applied a graph matching technique to identify patterns in UML class diagram of an application. Their technique categorized graph matching into K-phases. Pradhan et al. [18] have presented design pattern detection scheme that is based on graph isomorphism and normalized cross-correlation techniques.

Uchiyama et al. [19] have presented a software pattern detection approach by using software metrics and learning based method. They have identified candidates for the roles that compose the design patterns by considering machine learning and software metrics. Authors have considered backpropagation algorithm as a machine learning technique. Alhusain et al. [20] have proposed the pattern identification approach, which uses machine learning algorithm i.e., artificial neural network (ANN). Authors have developed a training dataset by using various existing pattern detection tools and subsequently they have applied feature selection technique for retrieving the input feature vectors.

Chihada et al. [21] have proposed software pattern identification process that is based on learning methods. They have considered support vector machine (SVM) for the classification of their selected design patterns. Yu et al. [22] have proposed a pattern detection technique using sub-patterns and method signatures concepts. Firstly, they have translated the source codes and Gamma et al. [1] patterns into graphs then identified the instances of sub-patterns. Alnusair et al. [23] have proposed ontology-based detection of software patterns. They have considered semantic representations of software patterns encoded with ontology constructs and SWRL rules.

III. PRELIMINARIES

Presented approach considers three approaches such as design patterns, software metrics and supervised learning methods, which are described as follows.

A. Selected set of design patterns

The proposed study considers five number of design patterns such as Abstract Factory, Adapter, Bridge, Singleton, and Template Method for the mining of software patterns available in source code of a project i.e., JUnit. Abstract Factory and Singleton belong to creational design pattern. Adapter and Bridge belong to structural design pattern. Template Method is a class of behavioral design pattern. Singleton design pattern ascertains to variety of classes in such as way that they create only one instance and support global point of access. Abstract Factory is often used to support an interface for generating a set of related objects without representing their concrete classes. A structural pattern i.e., Adapter helps to translate interface of a class into another. It is composed of four participants such as Adapter, Adaptee, Target, and User. Bridge design pattern helps to separate an abstract class from its concrete class so that they can vary independently. Template Method offers a framework of a procedure in a method, which defer few functions to client subclasses.

B. Selected Set of Software Metrics

A pattern-based solution is often used as a methodology in object-oriented (OO) system, which considers a good number of metrics for measuring quality of software developed. In this study, sixty seven number of metrics are considered for experimenting pattern mining process. These metrics of an open source software are extracted by using JBuilder tool [24]. The list of metrics are presented in Table I (where M stands for metrics).

TABLE I: List of selected metrics

Sl.	M.	Sl.	M.	Sl.	M.	Sl.	M.	Sl.	M
1	A	15	CIW	29	IUR	43	NOA	57	PC
2	AC	16	CM	30	LCOM3	44	NOAM	58	PF
3	AHF	17	COC	31	LOC	45	NOC	59	PIS
4	AID	18	CR	32	MDC	46	NOCC	60	PS
5	AIF	19	ChC	33	MHF	47	NOCP	61	PUR
6	AIUR	20	DAC	34	MIC	48	NOED	62	RFC
7	AALD	21	DD	35	MIF	49	NOIS	63	RMD
8	AOFD	22	DOIH	36	MNOB	50	NOLV	64	TCC
9	AUF	23	EC	37	MNOL	51	NOM	65	WCM
10	CA	24	FO	38	MPC	52	NOO	66	WMPC1
11	CBO	25	HDiff	39	MSOO	53	NOOM	67	WOC
12	CC	26	HEff	40	NAM	54	NOP		
13	CE	27	HPLen	41	NCC	55	NOPA		
14	CF	28	I	42	NIC	56	NORM		

The presented approach considers sixty-seven number of metrics from various object-oriented aspects such as class-level metrics, inheritance-level metrics, coupling-level metrics, cohesion-level metrics, etc. These metrics help to measure the properties of OO system. In this study, the selected set of metrics are used in reverse engineering approach for recognizing actual candidate classes available in source code of a software.

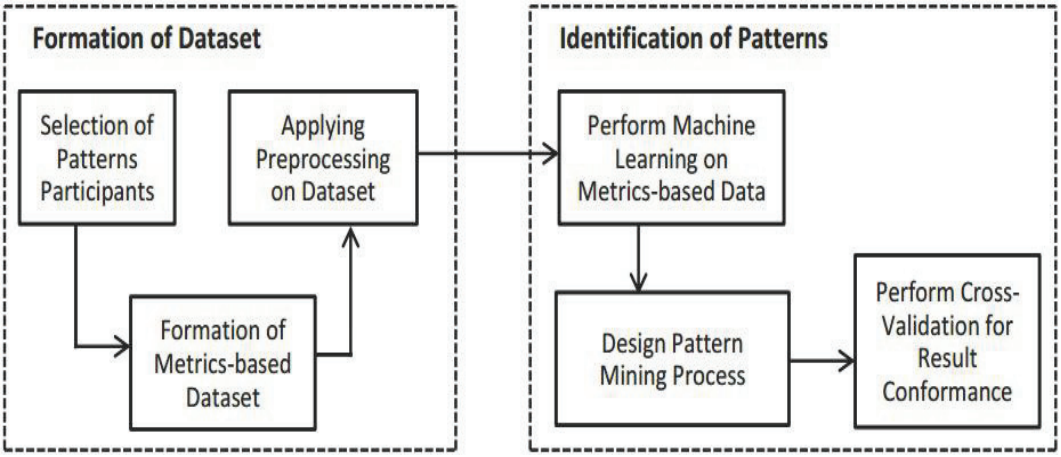

Fig. 1: The process of mining of software patterns

C. Learning Based Methods Applied for Pattern Mining

In this study, Layer Recurrent Neural Network (LRNN) and Random Forest are used as supervised learning methods for the mining of the prepared dataset. These methods help to build a model which provides predictions on the basis of known results. LRNN is an special form of an Artificial Neural Network (ANN) where it has a feedback loop having a delay around each layer of the network except for the last layer. It generalizes the ANN by adding a random number of layers and assigns random transfer functions for those layers. Random Forest is an ensemble learning technique for the classification of object-oriented metrics-based dataset. It works by creating a multitude of decision trees at training time and generating the classes of the individual trees. This classifier helps to reduce over-fitting problem of individual decision tree.

IV. Proposed Work

The presented approach is carried out in two phases, such as formation of pattern-oriented dataset and identification of software patterns as shown in Figure 1.

A. Formation of Dataset

In this phase, metrics-based pattern dataset is developed for classification of pattern instances. Formation of dataset includes three subprocesses such as selection of patterns participants, formation of metrics-based dataset and applying preprocessing on the dataset.

1) Selection of Patterns Participants: Design pattern notations are often represented by considering pattern template elements such as intent, problem, consequences, motivation, structure, related pattens, etc. These elements of pattern template help to system analyst for identifying right kind of pattern participant. In this study, five numbers of design patterns such as Abstract Factory, Adapter, Bridge, Singleton and Template Method are used for the process of pattern identification. These patterns have variety of participants. Single participant is associated with the Singleton pattern. Whereas four participants such as Abstract Factory, Abstract Product, Concrete Factory, and Concrete Product are associated with Abstract Factory pattern. Adapter pattern is associated with three number of participants such as Adapter, Adaptee, and Target. Four number of participants such as RefinedAbstraction, Abstraction, Implementor and ConcreteImplementor are associated with Bridge pattern. Template Method is associated with two numbers of participants such as AbstractClass and ConcreteClass.

2) Formation of Metrics-Based Dataset: Software metrics-based feature vectors can be developed by using design pattern detection tools such as similarity scoring algorithm [4], Web of Patterns [5], Metrics and Architecture Reconstruction PLugin for Eclipse [7]. These tools are used for the formation of metrics-based dataset where pattern instances are retrieved, which are common in more than one tools. Formation of metrics-based dataset is described by using a flowchart as shown in Figure 2. The formation of dataset is carried out by providing source code of an open source software i.e., JUnit as input to various software pattern detection tools, which extract pattern instances. Simultaneously, the source code is also given as input to JBuilder tool for extracting metrics-based candidate classes. Subsequently, metrics-based candidate instances are mapped with pattern instances extracted from various pattern detection tools. Finally feature vectors are created which include the metric values of all pattern's participants. For example, Abstract Factory pattern is associated with four numbers of participants and it includes total two hundred sixty eight ($67 \times 4 = 268$) number of feature vector for single instance of Abstract Factory pattern.

3) Applying Preprocessing on Dataset: Preprocessing of metrics-based dataset is carried out before learning process. In this process, whole dataset is of size 66×269 is categorized into input dataset of 66×268 and target dataset of size of 66×1. Eighty percent of metrics-based pattern dataset is considered for training of the model, whereas the remaining twenty percent data is used for testing the accuracy of the model.

B. Identification of Software Patterns

In this phase, learning process has been performed for the classification of pattern instances. During the pattern detec-

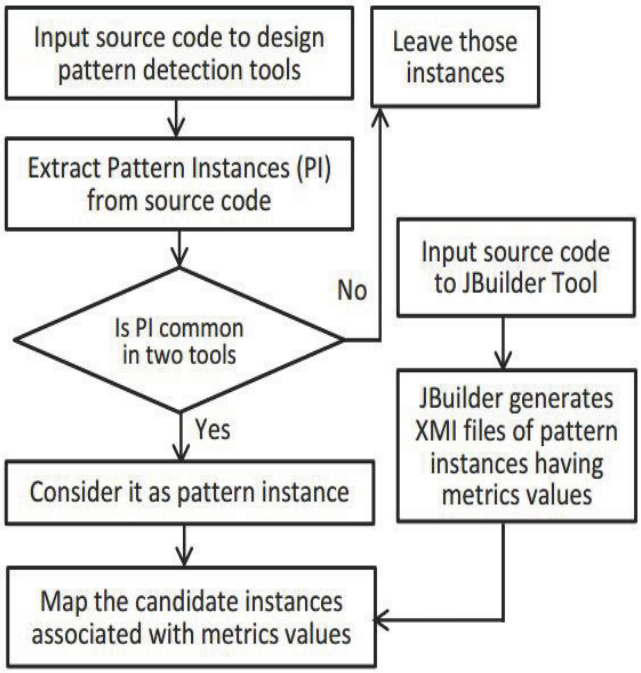

Fig. 2: Preparation of training dataset

tion process, it checks whether the composition of patterns available in source code are instances of the actual software patterns or not. The process of pattern identification includes three subprocesses such as learning of metrics-based dataset, design pattern mining process, and performing cross-validation for result conformance.

1) Learning of Metrics-Based Dataset: The presented study considers two classifiers such as Layer Recurrent Neural Network and Random Forest for the learning process. Unlike FFNN (Feed-Forward Neural Network), LRNN uses their internal memory for executing random number of inputs. Another reason for considering LRNN is that it is recurrent in nature. In the presented learning scheme, 66 × 268 feature vectors are considered as input and 66 × 1 feature vector is considered as output. Another classifier i.e., Random Forest is used as a predictive model for classifying input feature vectors on the basis of target values. The main advantage of this method is that it doesn't treat features as linear structure.

2) Process of Design Pattern Mining: During detection process, the composition of pattern participants available in training dataset are checked against trained pattern participants. LRNN and Random Forest are used for learning process.

3) Validation for Result conformance: Finally, validation of result is performed by using patterns definition as well as detection process.

V. Experimental Results

A set of experiments has been performed by considering an open source software i.e., JUnit for identifying pattern instances of the Abstract Factory, Adapter, Bridge, Singleton, and Template Method design patterns. JUnit software contains six instances of Abstract Factory, eleven instances of Adapter, nine instances of Bridge, two instance of Singleton, and thirty eight instances of Template Method design patterns. Hence, total sixty six number of pattern instances are identified from JUnit software. Subsequently, these pattern instances are categorized as training dataset as well as testing dataset in 80-20 ratio. For the critical examination of the presented study, P-MARt repository [25] is considered. It contains pattern instances of nine open source software, where one of them is JUnit.

In this study, the process of design pattern mining is measured by considering three performance parameters such as precision, recall, and harmonic mean of precision and recall i.e., F-measure as shown in Equation (1), (2), and (3) below:

$$Precision = \frac{\sum_{i=1}^{NDP} TP_i}{\sum_{i=1}^{NDP} TP_i + FP_i} \quad (1)$$

$$Recall = \frac{\sum_{i=1}^{NDP} TP_i}{\sum_{i=1}^{NDP} TP_i + FN_i} \quad (2)$$

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (3)$$

In equations, NDP denotes the number of patterns instances. True positive (TP) denotes the number of patterns available in training dataset and also classifier identifying them. False Positive (FP) denotes the number of patterns which are not available in training dataset but classifier identifying them. False negative (FN) denotes the number of patterns available in training dataset but classifier has not identified them. In this study, the value of F-measure is directly proportional to the accuracy result of software pattern identification.

The presented approach considers confusion matrix for depicting accuracy measurement parameters. Confusion matrices for LRNN and Random Forest are presented in Figure 3 and Figure 4 respectively. The overall accuracy for the selected design patterns are generated from trained classifiers such as LRNN and Random Forest, presented in Table II and Table III respectively. The presented tables show the values of true positive, false positive, false negative, precision, recall, F-measure, and accuracy for the set of design patterns. LRNN and Random Forest provide 100% values for F-measure for Singleton and Template Method. Random Forest provides 100% accuracy value for pattern mining process whereas LRNN provides 97% accuracy.

VI. Comparison

From the literature survey, it is observed that some of the existing approaches have considered manual tagging techniques for the preparation of dataset, which are time consuming processes. These processes may lead false positive rates. Presented approach considers some existing methodologies presented by Tsantalis et al. [4], Alnusair et al. [23], Chihada et al. [21], and Yu et al [22] for the evaluation of performance. The process of evaluation is made by examining the results of proposed study with the existing techniques for JUnit case study. Tsantalis et al. [4] have presented the value of

TABLE II: Pattern mining accuracy result obtained from LRNN

Software Patterns	True Positive	False Positive	False Negative	Precision	Recall	F-measure	Accuracy
Abstract Factory	5	1	0	83.3%	100%	90.9%	97%
Adapter	10	1	1	90.9%	90.9%	90.9%	
Bridge	9	0	1	100%	90.0%	94.7%	
Singleton	2	0	0	100%	100%	100%	
Template Method	38	0	0	100%	100%	100%	

TABLE III: Pattern mining accuracy result obtained from Random Forest

Software Patterns	True Positive	False Positive	False Negative	Precision	Recall	F-measure	Accuracy
Abstract Factory	6	0	0	100%	100%	100%	100%
Adapter	11	0	0	100%	100%	100%	
Bridge	9	0	0	100%	100%	100%	
Singleton	1	0	0	100%	100%	100%	
Template Method	38	0	0	100%	100%	100%	

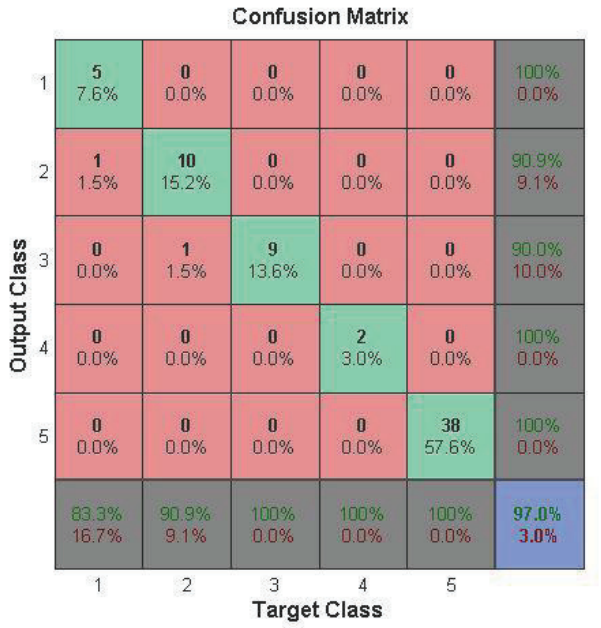

Fig. 3: Pattern detection accuracy generated from LRNN

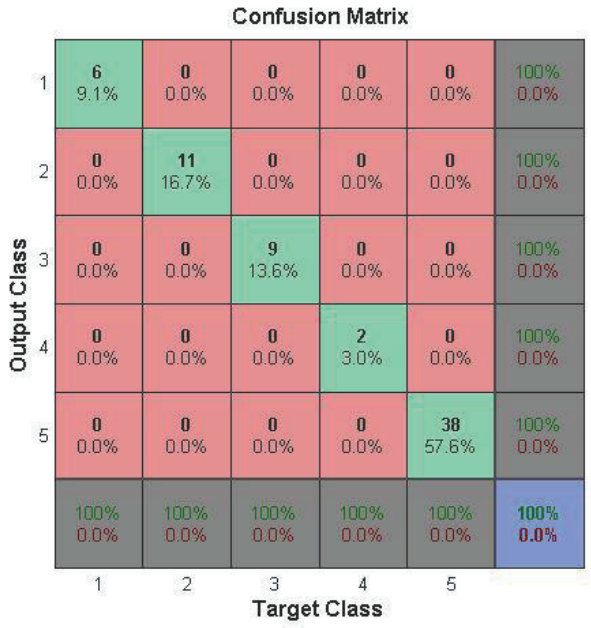

Fig. 4: Pattern detection accuracy generated from Random Forest

precision i.e., 17% and the value of recall i.e., 100% for Adapter pattern. they have shown 17% precision and recall for Template Method pattern. Alnusair et al. [23] have shown 100% precision and recall for Adapter and Template Method patterns. Chihada et al. [21] have shown results for Iterator and Observer patterns by using JUnit software. Yu et al. [22] have presented 100% precision and recall for Adapter, Singleton and Template Method patterns. In this comparative analysis, two observations have been obtained such as the presented approach offer better result for the process of pattern identification and most of the existing techniques have not presented pattern detection result for the Bridge pattern.

VII. Conclusion

Design pattern mining is considered as reverse engineering technique; it is helpful for the documentation and maintainability of the existing software. In this study, the process of design pattern mining is presented by considering learning based methods such as Layer Recurrent Neural Network and Random Forest. This study includes Abstract Factory, Adapter, Bridge, Singleton, and Template Method design patterns for the design pattern mining process. In order to evaluate the proposed approach, an open source project e.g., JUnit is taken into consideration for mining of selected patterns. The presented study considers object-oriented metrics for the dataset preparation process, which help to determine actual pattern instances. The benefits of using learning based methods for

the process of software pattern mining is that the proposed method uses object-oriented metrics-based dataset, which help to determine actual pattern instances.

The presented approach may be extended by identifying other patterns belong to variety of pattern catalog and available in large open source projects.

ACKNOWLEDGMENT

The presented work is supported by Web engineering and Cloud computing lab. A project is provided by DST, government of India.

REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[2] A. K. Dwivedi and S. K. Rath, "Incorporating security features in service-oriented architecture using security patterns," *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 1, pp. 1–6, 2015.

[3] D. Alur, D. Malks, J. Crupi, G. Booch, and M. Fowler, *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*. Sun Microsystems, Inc., 2003.

[4] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design pattern detection using similarity scoring," *Software Engineering, IEEE Transactions on*, vol. 32, no. 11, pp. 896–909, 2006.

[5] J. Dietrich and C. Elgar, "Towards a web of patterns," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 108–116, 2007.

[6] A. K. Dwivedi, A. Tirkey, and S. K. Rath, "An ontology based approach for formal modeling of structural design patterns," in *Contemporary Computing (IC3), 2016 Ninth International Conference on*. IEEE, 2016, pp. 208–213.

[7] M. Zanoni, F. A. Fontana, and F. Stella, "On applying machine learning techniques for design pattern detection," *Journal of Systems and Software*, vol. 103, pp. 102–117, 2015.

[8] A. K. Dwivedi and S. K. Rath, "Formalization of web security patterns," INFOCOMP Journal of Computer Science, vol. 14, no. 1, pp. 14–25, 2015.

[9] J. Dong, Y. Zhao, and T. Peng, "A review of design pattern mining techniques," *International Journal of Software Engineering and Knowledge Engineering*, vol. 19, no. 06, pp. 823–855, 2009.

[10] D. S. Kent Beck, Erich Gamma and M. Clark, "JUnit," http://junit.org/junit4/, 2014.

[11] Q. Liu and J. Wang, "A one-layer recurrent neural network with a discontinuous hard-limiting activation function for quadratic programming," *Neural Networks, IEEE Transactions on*, vol. 19, no. 4, pp. 558–570, 2008.

[12] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[13] Y.-G. Gueheneuc, H. Sahraoui, and F. Zaidi, "Fingerprinting design patterns," in *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*. IEEE, 2004, pp. 172–181.

[14] J. Dong, Y. Sun, and Y. Zhao, "Design pattern detection by template matching," in *Proceedings of the 2008 ACM symposium on Applied computing*. ACM, 2008, pp. 765–769.

[15] M. Gupta, A. Pande, R. Singh Rao, and A. Tripathi, "Design pattern detection by normalized cross correlation," in *Methods and Models in Computer Science (ICM2CS), 2010 International Conference on*. IEEE, 2010, pp. 81–84.

[16] A. S. Ba-Brahem and M. Qureshi, "The proposal of improved inexact isomorphic graph algorithm to detect design patterns," *arXiv preprint arXiv:1408.6147*, 2014.

[17] M. Gupta, R. Singh Rao, and A. K. Tripathi, "Design pattern detection using inexact graph matching," in *Communication and Computational Intelligence (INCOCCI), 2010 International Conference on*. IEEE, 2010, pp. 211–217.

[18] P. Pradhan, A. K. Dwivedi, and S. K. Rath, "Detection of design pattern using graph isomorphism and normalized cross correlation," in *Contemporary Computing (IC3), 2015 Eighth International Conference on*. IEEE, 2015, pp. 208–213.

[19] S. Uchiyama, H. Washizaki, Y. Fukazawa, and A. Kubo, "Design pattern detection using software metrics and machine learning," in *Joint 1st Int. Workshop on Model-Driven Software Migration, MDSM* 2011 and the 5th International Workshop on Software Quality and Maintainability, SQM 2011-Workshops at the 15th European Conf. on Software Maintenance and Reengineering, CSMR 2011, 2011.

[20] S. Alhusain, S. Coupland, R. John, and M. Kavanagh, "Towards machine learning based design pattern recognition," in *Computational Intelligence (UKCI), 2013 13th UK Workshop on*. IEEE, 2013, pp. 244–251.

[21] A. Chihada, S. Jalili, S. M. H. Hasheminejad, and M. H. Zangooei, "Source code and design conformance, design pattern detection from source code by classification approach," *Applied Soft Computing*, vol. 26, pp. 357–367, 2015.

[22] D. Yu, Y. Zhang, and Z. Chen, "A comprehensive approach to the recovery of design pattern instances based on sub-patterns and method signatures," *Journal of Systems and Software*, vol. 103, pp. 1–16, 2015.

[23] A. Alnusair, T. Zhao, and G. Yan, "Rule-based detection of design patterns in program code," *International Journal on Software Tools for Technology Transfer*, vol. 16, no. 3, pp. 315–334, 2014.

[24] CodeGear, "JBuilder," http://www.embarcadero.com/products/jbuilder, 2008.

[25] Y.-G. Guéhéneuc, "P-MARt: Pattern-like micro architecture repository," Proceedings of the 1st EuroPLoP Focus Group on Pattern Repositories, 2007.