Software Design Pattern Recognition using Machine Learning Techniques

Ashish Kumar Dwivedi[1], Anand Tirkey[2], Ransingh Biswajit Ray[3], Santanu Kumar Rath[4]
Department of Computer Science and Engineering
National Institute of Technology
Rourkela, 769008, Odisha, India
Email: shil2007@gmail.com[1], andy9c@gmail.com[2], ransingh.b.ray@gmail.com[3], skrath@nitrkl.ac.in[4]

***Abstract*—Design patterns helpful for software development are the reusable abstract documents which provide acceptable solutions for the recurring design problems. But in the process of reverse engineering, it is often desired to identify as well as recognize design pattern from source code, as it improves maintainability and documentation of the source code. In this study, the process of software design pattern recognition is presented which is based on machine learning techniques. Firstly, a training dataset is developed which is based on software metrics. Subsequently, machine learning algorithms such as Layer Recurrent Neural Network and Decision Tree are applied for patterns detection process. In order to evaluate the proposed study, an open source software i.e., JHotDraw_7.0.6 has been used for the recognition of design patterns.**

***Index Terms*—Software Design Patterns, Design Pattern Recognition, Machine Learning, Object-Oriented Metrics.**

I. Introduction

Software design pattern is a collection of techniques, which are used to exchange knowledge across a particular domain. A good number of software patterns are available in the literature for implementing a particular system [1] [2] [3]. These patterns are analyzed by using a number of tools and techniques [4] [5] [6] [7] [8]. 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] also known as Gang of Four (GoF) 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, structural and behavioral patterns. These software patterns also help to improve software maintenance by making explicit representation of class and object interactions.

The concept of design patterns is being accepted in the field of forward engineering, where a number of applications have been used patterns as a solution for their recurring design problems. A pattern-oriented solution is based on its candidate classes, which play variant roles during the construction of an application. Few patterns have similar structure such as State and Strategy patterns, but their candidate classes contain distinct roles. Therefore, it is essential to identify the right kind of role of pattern participants. These pattern's role facilitate the understandability of original design decisions of an application. Nowadays, the number of patterns increases continuously, which occur in a large amount of pattern's role. Hence, the recognition process may not be based on a particular pattern. The process also require the behavioral aspect of patterns.

Design patterns are often represented by using a semi-formal notation 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. A good number of formal modeling notations are available for analyzing pattern-based implementation template [6] [4] [9]. 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 [8]. Recognition of software design pattern is a reverse engineering technique; most of the times suffer from false positive as well as false negative issues [10]. These issues affect the accuracy of design pattern recognition process.

The presented approach helps to remove these problems by proposing a method, which translates the process of software design pattern recognition into a learning process. In this process, design pattern instances available in source code of an object-oriented system are classified by using supervised learning methods. In order to perform design pattern recognition process, classification techniques such as Layer Recurrent Neural Network (LRNN) and Decision Tree are applied. For the evaluation of the proposed approach open source software i.e., JHotDraw_7.0.6 [11] has been considered. Classification process is carried out by considering software design patterns such as Abstract Factory and Adapter patterns. It is observed that the learning based methods are often used to develop a model that helps to predict the outcomes of testing dataset after the completion of learning process. Testing dataset is mainly used to validate the trained model. It is observed that, the developed model with higher prediction accuracy may be gained by utilizing larger training dataset that can be generalized for new datasets. The proposed approach is based on supervised learning based classification algorithm, which uses object-oriented metrics-based dataset. Object-oriented software properties are often measured by using software metrics [12].

II. Related Work

A good number of design pattern detection techniques are available in the literature [10]. Various techniques are based on static analysis [5] [13] [14] facing problems when two or more patterns have same structure. This section presents an analysis of related techniques.

The concept of design pattern identification was firstly presented by Shull et al. [15]. Authors have applied manual checking of the presence of pattern instances, which becomes time consuming. Another manual tagging of pattern instances was performed by Gueheneuc et al. [16]. They 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 manually and developed a repository of identified instances known as design pattern library. Further they have applied machine learning algorithm to recognize classes playing certain roles.

Tsantalis et al. [5] have proposed an approach i.e., detection of design pattern, which uses similarity scoring algorithm on vertices of a graph that shows a design pattern graph. Authors have used similarity algorithm to identify software patterns, which correspond to a piece of code available in one or more inheritance hierarchies. Dong et al. [13] 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. Pradhan et al. [17] have applied two methods such as graph isomorphism and normalized-cross correlation for the identification of software design patterns.

Ferenc et al. [18] have applied machine learning techniques for the recognition of design patterns. They have considered backpropagation neural network and decision tree for detecting Adapter as well as Strategy patterns. Uchiyama et al. [19] have presented a software pattern detection approach by using software metrics and machine learning technique. 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 presented the process of design pattern detection, which is based on supervised learning methods. Authors have applied support vector machine (SVM) for the classification of selected set of pattern instances available in source code of open source projects. Gupta et al. [22] have presented a technique to identify software patterns by considering Normalized Cross Correlation while taking design pattern as a template to find its presence in the system design. Di Martino and Esposito [23] have presented an approach for the recognition of design patterns. They have used web ontology language (OWL) for pattern detection process.

III. Preliminaries

The proposed study includes various techniques such as design patterns, software metrics, learning based methods, etc., which require proper description.

A. Software Design Patterns

In this study, Abstract Factory and Adapter patterns are considered for identifying patterns from source code. Abstract Factory is a creational design pattern mainly used to offer an interface for creating families of related or dependent objects without specifying their concrete classes as shown in Figure 1 [1]. This pattern contains four participants such as AbstractFactory, ConcreteFactory, AbstractProduct, and ConcreteProduct. Adapter is a structural design pattern mainly used to convert the interface of a class into another interface. It contains three participants such as Adapter, Adaptee and Target as shown in Figure 2 [1].

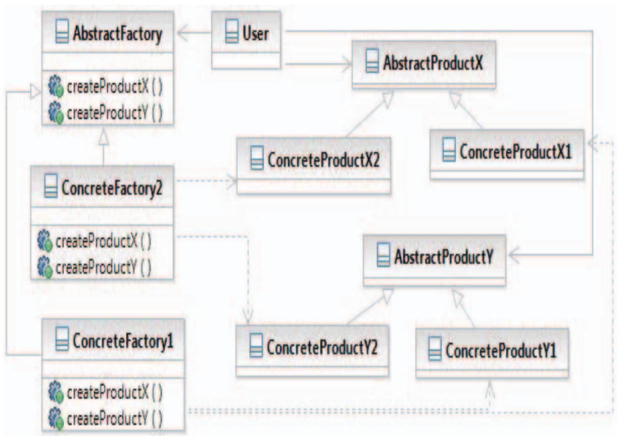

Fig. 1. Abstract Factory Design Pattern

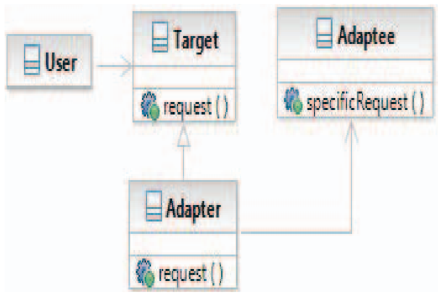

Fig. 2. Adapter Design Pattern

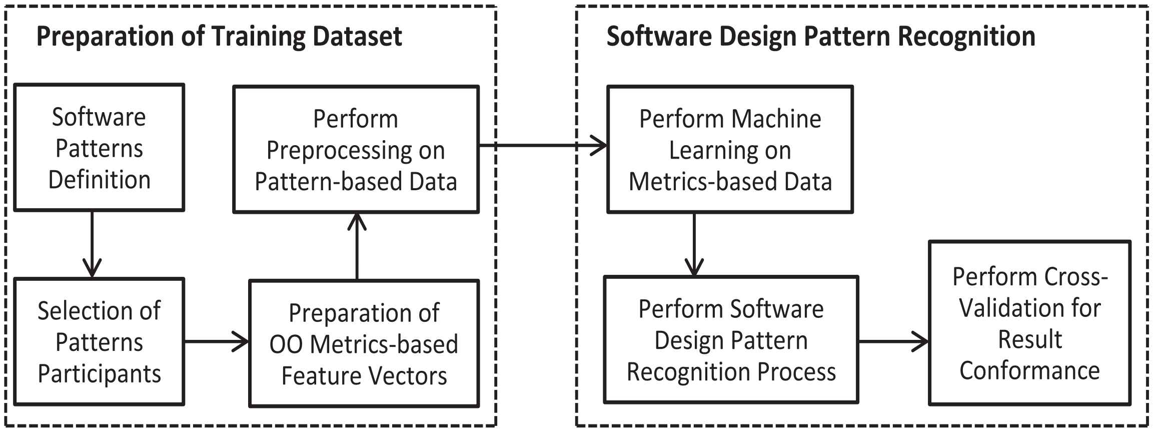
Preparation of Training Dataset
Software Patterns Definition
Selection of Patterns Participants
Preparation of OO Metrics-based Feature Vectors
Perform Preprocessing on Pattern-based Data
Software Design Pattern Recognition
Perform Machine Learning on Metrics-based Data
Perform Software Design Pattern Recognition Process
Perform Cross-Validation for Result Conformance

Fig. 3. Presented Software Design Pattern Recognition Model

B. Software Metrics used for the Creation of Feature Vectors

Design patterns are often used to provide solution for object-oriented (OO) system. The properties of these system can be measured by considering variety of OO metrics. The presented study includes sixty seven number of metrics for the recognition of software design patterns. The selected set of metrics are extracted by using JBuilder tool [24]. These software metrics are shown in Table I, where M stands for metrics.

TABLE I
LIST OF SELECTED METRICS

SN.	M.	SN.	M.	SN.	M.	SN.	M.	SN.	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		

C. Selected Supervised Learning Methods

The presented approach considers Layer Recurrent Neural Network (LRNN) [25] and Decision Tree [26] as supervised learning methods for the classification of the prepared dataset. The learning based algorithms are used to develop a model that provides predictions on the basis of available results. LRNN is a 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 an arbitrary number of layers and to have arbitrary transfer functions in each layer. Decision tree is considered to train the models by providing a classification tree of the datasets and predicting the outcomes of test data. In this tree, leaf nodes denote the outcomes.

IV. PROPOSED WORK

The presented approach is classified into two components such as preparation of pattern-based dataset and recognition of software design pattern as shown in Figure 3.

A. Preparation of Training Dataset

In this process, dataset is prepared for training the classifiers used in this study. Preparation of training dataset includes four other subprocesses such as software pattern definition, selection of patterns participants, preparation of OO metrics-based feature vectors, and perform preprocessing on pattern-based data.

1) Definition of Design Patterns: Generally design patterns can be described by using pattern template elements such as intent, problem, solution, related patterns, applicability, etc. The definition of software patterns is specified by considering pattern structure and behavior made by pattern specialists.

2) Selection of Patterns Participants: A design pattern is a collection of one or more number of classes, which are also known as pattern's participant. These participant classes have a particular role in a pattern-based system. These roles require unique identification during the recognition of design pattern. In this study, Abstract Factory and Adapter patterns are used where Abstract Factory contains four participants such as Abstract Factory, Abstract Product, Concrete Factory, and Concrete Product. Similarly, Adapter contains Adapter, Adaptee, and Target as pattern participants. Source code of an object-oriented system such as JHotDraw may contain role permutations of pattern participants. These role permutations can be minimized by removing classes which are not playing essential role.

3) Preparation of OO Metrics-Based Feature Vectors: Object-oriented metrics-based feature vectors can be prepared by using various software pattern detection tools such as similarity scoring algorithm [5], Web of Patterns [6], Metrics and Architecture Reconstruction Plugin for Eclipse [8]. Preparation of metrics-based dataset is presented in Figure 4. In this process, source code of an open source software i.e., JHotDraw is given as input to different pattern detection tools, which extract pattern instances. Simultaneously, the source code is also given input to JBuilder tool for extracting metrics based candidate classes. Subsequently, metrics-based candidate instances are mapped with pattern instances extracted from pattern detection tools. Finally feature vectors are created which include the metric values of all pattern's participant in a single row. For example, Abstract Factory pattern is associated with four numbers of participants and it includes two hundred sixty eight ($67 \times 4 = 268$) number of feature vectors for single instance of Abstract Factory.

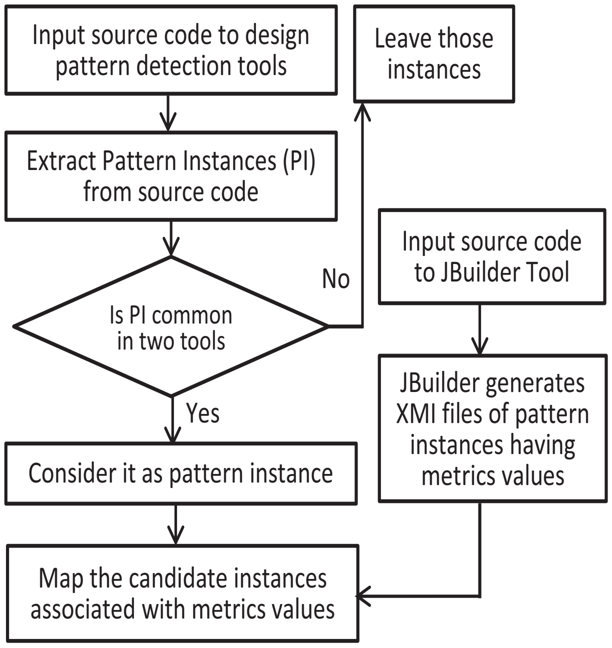
Input source code to design pattern detection tools
Leave those instances
Extract Pattern Instances (PI) from source code
Input source code to JBuilder Tool
No
Is PI common in two tools
JBuilder generates XMI files of pattern instances having metrics values
Yes
Consider it as pattern instance
Map the candidate instances associated with metrics values

Fig. 4. Preparation of training dataset

4) Preprocessing of Metrics-Based Dataset: Preprocessing of metrics-based dataset is carried out before learning process. In this process, whole dataset of size 219×269 is categorized into input dataset of size 219×268 and target dataset of size 219×1. The whole dataset is divided into eighty-twenty ratio, where eighty percent of dataset is used for the learning process, whereas other twenty percent dataset is considered for the testing process.

B. Recognition of Software Design Pattern

In this phase, binary classifiers are learned for the recognition of design patterns. It checks whether the composition of pattern participants available in source code are instances of the actual software patterns or not. The process of design pattern recognition contain three subprocesses such as learning of dataset, recognition of software patterns, and perform cross-validation for result conformance.

1) Learning of Metrics-Based Feature Vectors: The presented approach considers two classifiers such as Layer Recurrent Neural Network and Decision Tree 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. Decision tree is considered as a predictive model for classifying input feature vectors on the basis of target values. It helps in partitioning of training data in a recursive manner. Both LRNN and decision tree are discriminative models which are used for identifying explicit boundaries between classes.

2) Design Pattern Recognition Process: During the recognition of design pattern, the composition of pattern participants available in training dataset are checked against learned design pattern participants. In this process, LRNN and decision tree are used for the learning of dataset. In order to remove underfitting and overfitting from the selected models, five-fold cross-validation has been performed.

3) Validation for Result conformance: Finally, validation has been performed for the result conformance by using patterns definition presented by pattern specialist and obtained result from the design pattern recognition process.

V. Experimental Results

A set of experiments has been conducted by using an open source project i.e., JHotDraw for the recognition of software design patterns. JHotDraw software includes fifty nine instances of Abstract Factory and one hundred sixty instances of Adapter patterns. These pattern instances are categorized as training dataset as well as testing dataset as shown in Table II. During the experiment process, an evaluation of the proposed study is also performed by using a pattern repository i.e., P-MARt [27]. The repository contains pattern instances of nine open source software, where one of them is JHotDraw. In our approach, newer version of JHotDraw_7.0.6 is considered, which includes greater number of classes, whereas P-MARt repository uses JHotDraw_5.1.

TABLE II
Prepared Dataset before learning process

Software Patterns	**Number of instances**	**Training dataset**	**Testing dataset**
Abstract Factory	59	48	11
Adapter	160	128	32

In this study, three parameters such as precision, recall, and F-measure i.e., harmonic mean of precision and recall are considered to measure the accuracy of software design pattern recognition process. Precision, recall and F-measure are mathematically defined by using Equation (1), (2), and (3) respectively. In the presented equations, NDP denotes the number of patterns instances. True positive (TP) denotes the number of patterns available in training dataset and also classifier recognized them. False positive (FP) denotes the number of patterns which are not available in training dataset but classifier recognized them. False negative (FN) denotes the number of patterns available in training dataset but classifier has not recognized them. In the presented approach, the value of F-measure is directly proportional to the accuracy result of design pattern recognition process.

$$Precision = \frac{\sum_{i=1}^{NDP} TP_i}{\sum_{i=1}^{NDP} TP_i + FP_i} \tag{1}$$

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

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{3}$$

The presented approach has been used confusion matrix for depicting the accuracy measurement parameters. Confusion matrices for LRNN and decision tree are presented in Figure 5 and Figure 6 respectively. The overall accuracy for Abstract Factory pattern and Adapter pattern are generated from trained classifiers such as LRNN and Decision Tree, which are presented in Table III and Table IV respectively. These tables represent both training accuracy as well as testing accuracy. Both the classifiers provide 100% accuracy for the Adapter design pattern. LRNN provides 100% training and testing accuracy for the Abstract Factory pattern whereas decision tree provides 99.4% training accuracy and 97.7% testing accuracy for Abstract Factory pattern.

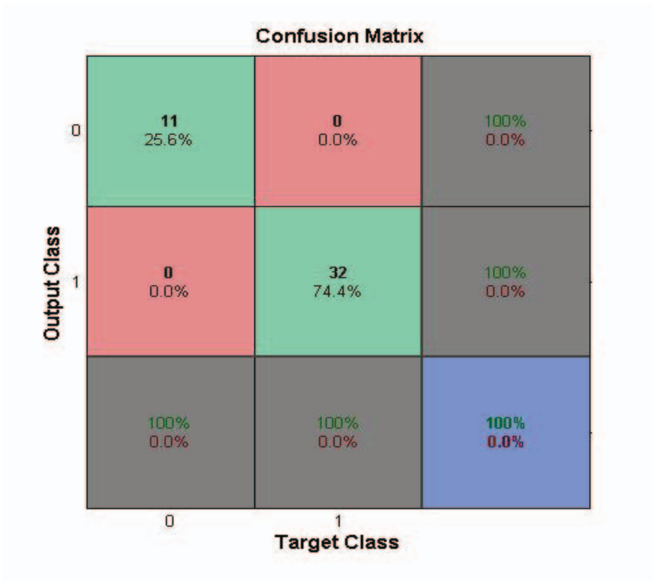

Fig. 5. Confusion matrix generated from LRNN

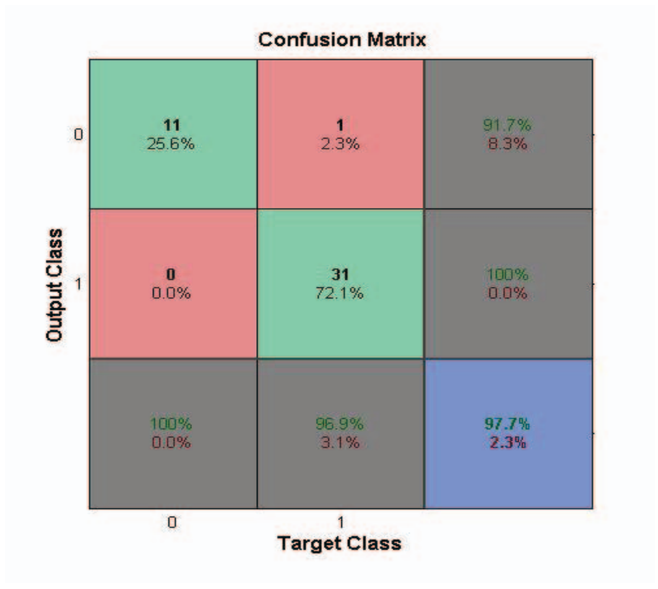

Fig. 6. Confusion matrix generated from Decision Tree

TABLE III
ACCURACY RESULT GENERATED FROM LRNN

Software Patterns	Precision	Recall	F-measure	Train Accuracy	Test Accuracy
Abstract Factory	100%	100%	100%	100%	100%
Adapter	100%	100%	100%	100%	100%

TABLE IV
ACCURACY RESULTS GENERATED FROM DECISION TREE

Software Patterns	Precision	Recall	F-measure	Train Accuracy	Test Accuracy
Abstract Factory	100%	91.7%	95.6%	99.4%	97.7%
Adapter	100%	100%	100%	100%	100%

VI. COMPARISON OF PERFORMANCE

In the literature, most of the existing techniques are based on manual tagging of dataset, which are time consuming process and it may lead false positive rate. The presented study performs evaluation of performance by comparing the results of proposed study with the existing techniques such as Ferenc et al. [18], Tsantalis et al. [5], Uchiyama et al. [19], and Chihada et al. [21]. Ferenc et al. have applied machine learning methods such as ANN and decision tree for the detection of Adapter as well as Strategy patterns and achieved 66.7% testing result. However they have not used JHotDraw as a case study. Tsantalis et al. have applied similarity algorithm for detecting design patterns. They have achieved 48% precision, 100% recall and 65% F-measure for JHotDraw. Uchiyama et al. have performed training of data based on OO metrics. They have achieved 100% precision and 90% recall for small scale program by considering backpropagation neural network as a classifier. Chihada et al. have performed pattern detection process by using support vector machine (SVM). They have achieved 86% precision, 86% recall and 86% F-measure for Adapter design pattern using JHotDraw. During the process of comparison, it is observed that our approach yields significantly better values of precision, recall and F-measure for the selected patterns.

VII. Conclusion

The presented design pattern recognition approach is carried out by using learning-based algorithms such as LRNN and Decision Tree. This approach considers sixty seven number of object-oriented metrics for the preparation of metrics-based dataset. The process of design pattern recognition is performed by using Abstract Factory and Adapter design patterns. In order to perform the experiment process, an open source software e.g., JHotDraw_7.0.6 has been used. The proposed study considers a technique i.e., preprocessing of dataset, which yields higher accuracy by minimizing the number of candidate patterns. The advantage of presented approach is the use of object-oriented metrics and machine learning techniques, which help to identify right kind of roles for the pattern participants and develop learning models, which improve quality parameters for the recognition of design patterns.

The proposed study can be extended by recognizing other software design patterns available in various open source object-oriented system.

Acknowledgment

Authors hereby acknowledge sincere thanks to the sponsorer of the Web engineering and Cloud computing lab. Project sanctioned by FIST programme, Department of Science and Technology, Govt 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] M. Fowler, *Patterns of enterprise application architecture*. Addison-Wesley, Boston, USA, 2002.

[4] H. Zhu and I. Bayley, "On the composability of design patterns," *Software Engineering, IEEE Transactions on*, vol. 41, no. 11, pp. 1138–1152, 2015.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] W. Randelshofer, "JHotDraw," https://sourceforge.net/projects/jhotdraw/, 2011.

[12] E.-M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "Software metrics fluctuation: a property for assisting the metric selection process," *Information and Software Technology*, vol. 72, pp. 110–124, 2016.

[13] 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.

[14] A. Blewitt, A. Bundy, and I. Stark, "Automatic verification of design patterns in java," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. ACM, 2005, pp. 224–232.

[15] F. Shull, W. L. Melo, and V. R. Basili, "An inductive method for discovering design patterns from object-oriented software systems," University of Maryland, Tech. Rep. UMIACS-TR-96-10, 1998.

[16] 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.

[17] 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.

[18] R. Ferenc, A. Beszedes, L. Fülöp, and J. Lele, "Design pattern mining enhanced by machine learning," in *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*. IEEE, 2005, pp. 295–304.

[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] 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.

[23] B. Di Martino and A. Esposito, "A rule-based procedure for automatic recognition of design patterns in uml diagrams," *Software: Practice and Experience*, 2015.

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

[25] 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.

[26] L. Rokach and O. Maimon, *Data mining with decision trees: theory and applications*. World Scientific Pub Co Inc. ISBN 978-9812771711, 2008.

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