2013 IEEE 25th International Conference on Tools with Artificial Intelligence

Design Pattern Recognition by Using Adaptive Neuro Fuzzy Inference System

Sultan Alhusain and Simon Coupland
Centre for Computational Intelligence
De Montfort University
Leicester, United Kingdom
Email: sultanib@dmu.ac.uk

Robert John
Automated Scheduling
Optimisation and Planning (ASAP) Group
University of Nottingham
Nottingham, United Kingdom

Maria Kavanagh
Business Development Manager
Emerald Hill Limited
Leicester, United Kingdom

***Abstract*—Software design patterns describe recurring design problems and provide the essence of best practice solutions. It is useful and important, for various software engineering tasks, to know which design pattern is implemented where in a software design. However, this information is often lost due to poor or absent documentation, and so accurate recognition tools are required. The problem is that design patterns, given their abstract and vague nature, have a level of resistance to be automatically and accurately recognized. Although this vagueness or fuzziness can be captured and modelled by the fuzzy inference system, it has not yet been applied to solve this problem. This paper fills this gap by proposing an approach for design pattern recognition based on Adaptive Neuro Fuzzy Inference System. Our approach consists of two phases: space reduction phase and design pattern recognition phase. Both phases are implemented by ANFIS. We evaluate the approach by an experiment conducted to recognize six design patterns in an open source application. The results show that the approach is viable and promising.**

***Keywords*-Software design patterns; machine learning; pattern recognition; ANFIS.**

I. Introduction

The idea of capturing and documenting experiences in the form of design patterns was proposed by the architect Christopher Alexander. A pattern, as described by him, is a description of a recurring design problem associated with the essence of its solution which can be applied "a million times over without doing it the same way twice" [1]. The desire to improve the software quality has led to the enthusiastic adoption of the design pattern principle in the programming community [2]. The proper application of a software design pattern (DP) helps to improve the quality because it is a means by which good design experiences can be reused instead of being rediscovered [3]. In fact, any high-quality software is likely to have occurrences of DPs in its design even if they are not intentionally introduced [4].

Software DPs can be defined as a recurring set of classes which are organized in a particular way for the purpose of solving a recurring design problem [5]. This recurring pair of a design problem and its solution is normally given a descriptive name. DP descriptions usually include UML class diagrams in which classes represent roles, and each role is defined in terms of a set of related responsibilities. When a DP solution is realised and applied in a software design, a DP occurrence or instance is said to be created.

The information about which DP is implemented facilitates various software engineering tasks, including software comprehension, maintenance and evolution as well as quality evaluation [6] [7]. However, this information is usually lost because software applications often have poor, obsolete or even no documentation at all [8]. The need to recover up-to-date documentation has raised the importance of DP recognition tools. So, the research field of DP recognition is currently very active [9].

When a DP instance is recognised, information about the responsibilities and roles of several classes is made available. Also, the original design problem, for which the DP is used, will be exposed. Providing such valuable information to maintainers enables them to make faster and well-informed maintenance decisions. This consequently helps to reduce the cost of the software maintenance, which is the most expensive stage in the software life cycle. [10].

The problem is that DPs are vaguely and abstractly defined, which gives them a level of resistance to automatic recognition. To make the matter worse, these abstract definitions are rarely followed strictly [7]. In fact, the DPs descriptions are theoretical and are not necessarily reflected accurately in the implemented instances [11]. Also, the boundaries of what does and does not constitutes a valid instance of a DP is "seldom or never completely clear-cut" [12]. So, the best approach to recognise DPs may be the use of *fuzzy* rules in fuzzy inference systems (FIS). Despite some claims, which will be disputed in the next section (II), the FIS has not been utilised wel to recognize software DPs.

In this paper, we describe an approach for DP recognition based on Adaptive Neuro Fuzzy Inference System (ANFIS) [13]. Using ANFIS makes it possible not to pre-set rules based on theoretical descriptions as it provides a systematic approach to learning them from training data. Our approach consists of two phases, both of which are implemented by ANFIS (fig. 1). The first phase is a search space reduction phase, in which a set of candidate classes is identified for each role in every DP. This phase is also expected to improve the accuracy of the overall system by removing classes which are certainly not playing any role in any

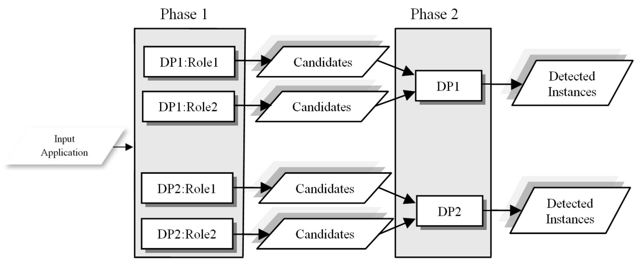

Figure 1. The overall design pattern recognition process.

DP. The task of the second phase is to recognise which of the related roles' candidates do actually constitutes a valid instance of a DP. To evaluate our approach, an experiment is conducted to recognize six DPs (Adapter, Command, Composite, Decorator, Observer and Proxy) in the JHotDraw 5.1[1] open source application.

The rest of this paper is organized as follows. In the next section (II), some of the existing DP recognition approaches is discussed, with a focus on those that implement artificial intelligence (AI) techniques. Section III discusses the model structure and training. In section IV, the two phases of DP recognition process is discussed in more detail. We report the evaluation results in Section V. The conclusions and future work are presented in the last section (VI).

II. Related Work

Although many approaches have been proposed for DP recognition, only a few of them used AI techniques. To the best of the authors' knowledge, AI related techniques have only been used to reduce the search space or to filter out false positives, as will be discussed below. The only exception is the author's previous paper which presented a recognition approach based on a machine learning technique [14]. Our previous work was the first that did not pre-set or derive any rules or features from the theoretical descriptions of DPs. Alternatively, feature selection methods were used to select the input feature vectors of artificial neural networks (ANNs) which were trained to perform the task of pattern recognition.

A DP recognition approach based on graph matching was proposed in [15]. In this approach, DPs were defined hierarchically in terms of sub-patterns, and they were both formally defined as graph transformation rules. Each rule was assigned a "fuzzy belief" value to express its probability of producing a correct match. Wherever a rule of a DP held true, an annotated node was created to represent the existence of the DP. The created nodes were associated with "fuzzy values" expressing their probability of being correct matches. Each fuzzy value was calculated as the minimum of the fuzzy beliefs of a rule and its sub rules. However, since the fuzzy values represent events (i.e. a rule node represents a correct match) which are partially *dependent* on whether the sub-rules have produced correct matches or not, they should have been calculated by the conditional probability formula and not by the minimum (compositional) function (i.e. $P(A \wedge B)$, when event A is dependent on event B, is not a function of $P(A)$ and $P(B)$). Although this approach is clearly based on probability and does not apply any FIS, it is claimed to be a "fuzzy logic based" [16]. Moreover, this approach confuses the concept of "degree of truth" with the concept of "degree of belief". The truth in the former is a matter of degree, which is what is meant to be captured by fuzzy logic, while it remains binary in the latter (e.g. 50% probability of being correct does not mean 50% correct as the ultimate truth is still either correct or incorrect) [17].

Another graph matching based approach was proposed in [7]. The recognition process in this approach includes two main phases. First, crisp logical rules were set based on the theoretical description of DPs to identify a set of candidate classes for each role in every DP. Then, to find DP instances, each combination of candidate classes were tested. To improve the recognition accuracy of their approach, decision trees and also ANNs were used to filter out false positives [4]. A linguistic value was suggested for most of the metrics used in the input feature vectors (e.g. "zero or a *low* value"), which could have been more appropriately modelled by fuzzy logic.

Another recognition approach, which was also based on graph matching, was introduced in [18]. Graph matching techniques were first applied to identify DP candidates. Then, classifiers implemented by several learning algorithms were trained to filter out false positives.

The approach proposed in [19] employed a rule learner to infer rules that can be used to reduce the search space. This was done by calculating the same set of 13 metrics for classes playing different roles. Then, the rule learner was applied to infer rules for each individual role. The inferred rules can then be used to identify a set of candidate classes for each role.

To reduce the search space, the approach proposed in [20] used only one ANN to recognize candidates for all roles in all DPs. However, the input feature vector of one class can sometimes have multiple target outputs because it can play multiple roles in different DPs. This multi-label classification problem was apparently ignored in this approach.

III. Model Structure and Training

A. Training Dataset Preparation

There is a recognized lack of DP standard benchmarks which can be used to train machine learning models, or even to evaluate other recognition approaches [12]. So, there is a need to overcome this drawback by preparing a training dataset. The method chosen to prepare the training dataset is based on using multiple existing DP recognition tools. Following this method, DP instances are added as

[1]http://www.jhotdraw.org

positive training examples to the training dataset only if they are detected by a minimum of two tools (i.e. received a minimum of 2 votes), while instances with one and only one vote are added as negative training examples.

This method is based on the assumption that the more votes a DP instance has, the more likely it is to be a true positive, and vise versa. Although that instances with zero votes are even more likely to be false positives than those with one vote, they are characteristically further apart from the positive training examples. The characteristics that enable instances with one vote (negative examples) to *deceive* one recognition tool are the ones that make them closer to those with two votes (positive examples) . So, drawing the classification boundary between the positive training examples and their closest negative ones can probably improve the accuracy of the DP recognition system. This idea is visually illustrated in fig. 2.

A training dataset has been prepared following this method by using 5 voters and over 400 open source applications from a wide range of application domains. The process of preparing the dataset, including the tools and the open source applications used, is discussed in more detail in our previous paper [14].

B. Features and Feature Selection

The feature sets, which should be used for DP recognition purposes, have not yet been researched [8]. To the best of our knowledge, and apart from our previous work, almost all previous approaches use sets of features derived from the theoretical descriptions of DPs. The only exception is the space reduction approach proposed in [19], which used a fixed set of class external quality attributes (e.g. coupling). The problem with theoretically derived features is that they are limited to features which are not validated on real implemented instances.

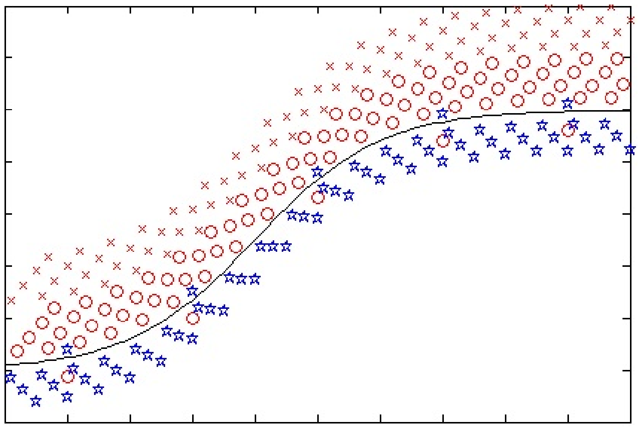

Figure 2. An ideal example of the classification boundary, where stars represent the instances with two or more votes, circles represent the instances with exactly one vote, and crosses represent the instances with zero votes.

To avoid such limitation, an extended set of features is compiled from features used in previous approaches (e.g. [19], [20], [21] and [22]) as well as many object oriented metrics (e.g. [23] and [24]). The features are then calculated for all instances in the dataset by using several tools. The tools are CKJM[2], JMT[3], POM[4] and Dependency Finder[5] as well as the underlying tool developed by [21], which is modified to generate numbers instead of Boolean values in order to provide more information to the classifiers. After calculating all features for all roles/DPs, selection methods are used to select a different feature subset for each one.

There are two broad types of feature selection methods: *filters* and *wrappers* [25]. In the filter methods, the relevancy of feature sets are evaluated independently of any classification model, while in the wrapper methods, they are evaluated based on the performance of the classifier to be used. Our approach implements a hybrid selection method in which a wrapper layer is placed on top of a filter layer.

In the filter layer, several selection methods (reliefF [26], fisher [27], Gini index [28] and spectrum [29]) are used to evaluate and rank each individual feature, which generate a set of four differently ordered feature sets for every role/DP. Then, in the wrapper layer, the feature sets of all roles/DPs are evaluated by constructing, training and testing multiple FISs for each set, each time with a different subset size. The subset which yields the best performance is the one finally selected. The performance evaluation is discussed later.

C. Model Construction

It is not always possible to discern how exactly a FIS should be structured in order to reflect the behaviour of the system to be modelled. This is the case in the DP recognition problem in which the type and shape of membership functions, and the rules that govern input/output mapping, are not completely known beforehand. The construction of FISs in such cases has been made possible by ANFIS, which provides a systematic approach to learn the model structure from a given input/output dataset. ANFIS adjusts and fine-tunes the parameters of membership functions as a mean by which fuzzy systems can learn from the datasets they are intended to model [13].

ANFIS is basically a FIS implemented in an adaptive network framework in which the layers correspond to the inference steps in Sugeno-type FIS, as shown in fig. 3. For the training to take place in ANFIS, an initial FIS has to be constructed first. This can be done by grid partitioning an input space with n input features into a predetermined number (m) of membership functions. However, this can generate too many fuzzy rules (i.e. m^n rules) which are too computationally expensive to train. So, as a less expensive

[2]http://www.spinellis.gr/sw/ckjm
[3]http://jmt.tigris.org
[4]http://wiki.ptidej.net/doku.php?id=pom
[5]http://depfind.sourceforge.net

alternative, the initial FIS is constructed by using subtractive clustering, which is a fast algorithm to find a set of initial clusters [30]. The centres of the identified clusters are actual data points which are assumed to be representing the behavior of the system to be modelled. These clusters can then be used as an initial set of fuzzy rules, which can later be optimized by using ANFIS.

The degree (μ) to which a fuzzy rule is satisfied is defined based on the distance between an input vector (x) and the values in its corresponding dimensions of a cluster centre (x_c), and it is calculated as follows:

$$\mu = e^{-\alpha \parallel x - x_c \parallel^2}$$

where α is a constant calculated based on a radius parameter, which determines the range of influence of a cluster center (the radius is set to 0.5 in this paper).

D. Model Training

After an initial FIS is constructed, it can be optimized by using ANFIS, which employs a hybrid learning algorithm. The parameters of antecedent membership functions in layer 1 (fig. 3) are tuned by using the gradient descent back-propagation algorithm, while the least-squares estimation method is used to identify the parameters of the consequent linear functions in layer 4. The overall output (y) of the FIS can then be calculated for any input vector (x) as follows:

$$y = \frac{\sum_{i=1}^{c} \mu_i \, f_i(x)}{\sum_{i=1}^{c} \mu_i}$$

where c is the number of rules and f_i is the consequent linear function of the i^{th} rule.

A separate FIS is constructed and trained for each role/DP to be recognized by using their respective training dataset. However, the prepared training datasets have much more negative training examples than positive ones. So, to avoid the problems caused by having imbalanced datasets [31], it is necessary to use an over-sampling technique to generate more balanced datasets. The technique used is the adaptive synthetic (ADASYN) sampling technique [32]. Unlike some other synthesizing techniques, ADASYN considers the neighbors of minority samples to identify and generate more synthetic samples for the "seed" ones. Also, the total number of the synthetic samples to be generated is calculated based the difference between the majority and minority training examples, which results in a more balanced dataset.

For training, each dataset is divided into training, checking and testing subsets, each of which has roughly the same proportion of positive and negative training examples. The training subset is used for the optimization of model parameters, while the checking subset is used to ensure that the model does not overfit the training data. The testing subset is then used to evaluate the generalization performance of

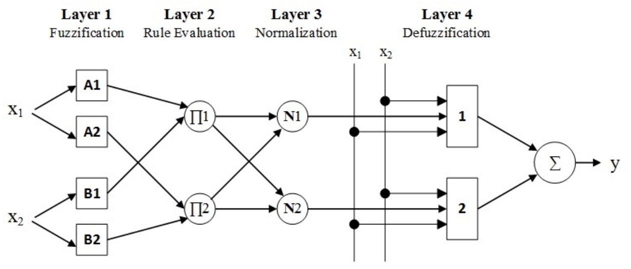

Figure 3. ANFIS architecture.

the trained models, which is measured by using the mean squared error (MSE). Since there are many possible feature subsets for each role/DP, a FIS is trained and tested for each subset. The FISs with the subsets that yield the least MSE are the ones used in the experiment reported in section V.

IV. The Design Pattern Recognition Process

A. Phase One (Intra-Class Level)

Phase one has two purposes. The first purpose is to reduce the search space by assigning a set of candidate classes to each role in every DP. Reducing the search space is expected to improve the efficiency of the proposed system. The second purpose is to improve the overall accuracy of the recognition system by removing classes that are definitely not playing any role in any DP. So, if too many false positive candidates are identified, the benefits of phase one will be compromised. However, on the other hand, the recall of the overall recognition system will be negatively affected if some true candidates are not recognized. So, a reasonable compromise should be struck and it should favour a high recall at the cost of low precision.

DPs are usually defined in terms of multiple classes playing different roles. Some of these roles are key and characteristically unique while others are not. In the field of DP recognition, "the occurrence type" denotes the set of roles recognized by a recognition technique and "the occurrence size" denotes the number of the roles recognized [12]. In our approach, we adopt the same occurrence as the one used in [21], which does not include the non-key roles as they would result in too many false positives, as they suggested. The experiments reported in [19] and [11] also showed that a low recall ratio was associated with the rules learnt for non-key roles, which were described as not having a fingerprint or a numerical signature. Nevertheless, it is easy to identify non-key roles manually after recognizing the key ones as they are usually closely related. Table I shows the roles considered for the DPs included in this paper.

The input feature vectors for each of the FISs constructed in phase one is a subset of 82 features capturing different aspects and properties at the intra-class level.

B. Phase Two (Inter-Class Level)

All of the recognized candidate classes in phase one are passed into phase two where each pair of related roles' candidates is examined whether it constitutes a valid instance of a DP or not. Without reducing the search space, this phase is likely to suffer from the combinatorial explosion problem as it would check all possible combinations of all classes in a software system.

The input feature vectors for each of the FISs constructed in phase two is a subset of 112 features. These features capture inter-class level relationships but also include some intra-class level features. The reason why some intra-class level features are again used in this phase is to capture any information hidden in the relativity between intra and inter class features. For example, in [4], the ratio between the number of inherited public methods in a class which invokes an external method (inter class feature) and the total number of public methods in the class (intra class feature) was found to be useful in filtering out false positives.

V. Evaluation Results

The objective evaluation of DP recognition approaches is known to be a challenging task, given the absence of any standard benchmarks [33]. Most of the approaches proposed in the literature have been evaluated by manually checking their results. However, such evaluation can be subjective as the same instance can be labelled differently by different evaluators, which may affect the validity of the evaluation. So, to evaluate our approach objectively, no instance is labelled as correct or incorrect based on our own judgment. Alternatively, the only available peer reviewed repository (PMARt [34]) is set as a main evaluation reference. Because this repository is not claimed to be complete, the relevant literature and the internal documentation of the application under analysis are also considered during the evaluation.

JHotDraw 5.1 is an open source java application which is designed based on some well-known DPs, which makes it an ideal and popular choice for the evaluation of proposed DP recognition approaches. Because of this, and also because it is included in the PMARt repository, it is chosen as the software on which our approach is evaluated. In the experiment reported in this paper, *MATLAB Fuzzy Logic toolbox* is used to construct, train and test the FISs.

Precision and recall, which are common information retrieval accuracy measures, are calculated for the FISs of all roles and DPs. The precision shows how much of the retrieved instances are relevant (i.e. true positives), while the recall shows how much of the relevant instances (i.e. all existing instances) are retrieved. Similarly to some other approaches, DP instances with common classes are counted as different instances and not aggregated into one.

In table I, which shows the evaluation results of the first phase, it can be seen that five of the roles have perfect recalls and four of which also have high precisions. Since the main purpose of phase one is to reduce the search space, it can be said that it has successfully fulfilled its purpose as it has reduced the search space by about 88% on average (JHotDraw 5.1 has 173 classes). Moreover, the achieved recall is reasonably high at 84% on average, which means that phase one does not greatly affect the recall of the overall system negatively.

The results of the second phase are presented in table II. The table shows that the Decorator DP is perfectly recognized. Although the precision of this phase has been negatively affected by the high number of false positives recognized for only one DP (the Observer), it still has a precision of 45% on average. The poor performance achieved for the Observer is probably caused by the poor quality of its training data. If this is the case, the performance can be improved by using more voters in the dataset preparation step, and then increasing the required minimum number of votes accordingly. The same may also apply to the Adapter and Command DPs. Nevertheless, the recall of this phase is reasonably high at 75.21% on average.

Comparing different recognition tools based on how many instances they share with PMARt can represent a useful indication. Despite the fact that it is not a complete repository, it is still the only peer reviewed one. A comparison between our approach and three other approaches is presented in tables III and IV. It can be seen in table III that our approach

Table I
Phase One Evaluation Result

DPName:RoleName	***TP***	***FP***	***FN***	***Precision***	***Recall***
Adapter:Adaptee	7	25	4	21.88	63.64
Adapter:Adapter	20	38	7	34.48	74.07
Decorator:Component	3	0	0	100	100
Decorator:Decorator	4	1	0	80	100
Observer:Subject	2	39	1	4.88	66.67
Observer:Observer	2	12	0	14.29	100
Proxy:RealSubject	0	11	0	0	100
Proxy:Proxy	0	0	0	100	100
Composite:Component	1	0	0	100	100
Composite:Composite	3	9	2	25	60
Command:Receiver	3	4	1	42.86	75
Command:ConcreteCommand	9	41	4	18	69.23

Table II
Phase Two Evaluation Result

DP Name	***TP***	***FP***	***FN***	***Precision***	***Recall***
Adapter	4	31	22	11.4	15.4
Decorator	4	0	0	100	100
Observer	2	213	1	0.90	66.7
Proxy	0	0	0	100	100
Composite	5	7	0	41.7	100
Command	9	45	4	16.7	69.2

Table III
PHASE ONE COMPARSION BASED ON PMART

DP Name	**PMARt**	**Our Approach**		**SSA** [21]		**MARPLE** [18]		**FINDER** [35]	
	Detected	***Detected***	***Shared***	***Detected***	***Shared***	***Detected***	***Shared***	***Detected***	***Shared***
Adapter:Adaptee	**8**	31	6	13	3	30	6	1	0
Adapter:Adapter	**21**	58	17	21	3	46	5	1	0
Decorator:Component	**1**	3	1	3	1	14	1	1	1
Decorator:Decorator	**1**	5	1	3	1	21	1	2	1
Observer:Subject	**2**	41	2	2	0	38	2	10	0
Observer:Observer	**2**	14	2	3	1	21	2	5	0
Proxy:RealSubject	**0**	11	0	0	0	0	0	0	0
Proxy:Proxy	**0**	0	0	0	0	0	0	0	0
Composite:Component	**1**	1	1	1	1	5	1	4	1
Composite:Composite	**5**	12	3	1	1	9	3	9	5
Command:Receiver	**4**	7	3	13	3	116	4	2	2
Command:ConcreteCommand	**13**	50	9	21	8	111	13	7	5

shares more role-playing classes than the other approaches. Although that our approach shares 19 instances in phase two (table IV) compared with 21 shared by MARPLE, our approach has approximately 60% less unshared instances. Some unshared instances may actually be true positives, but they are more likely to be false positives. So, overall, the results shows that our approach is a viable and promising approach for DP recognition.

VI. CONCLUSIONS AND FUTURE WORK

We have described a two phase DP recognition approach based on ANFIS. The task of the first phase is to recognize a set of candidate classes for each roles in every DPs. The DPs are then recognized in the second phase. The fuzzy recognition rules are completely learned from training datasets and are not set based on DP theoretical descriptions. The initial FISs are constructed by using subtractive clustering and then trained by ANFIS.

To evaluate the proposed approach, an experiment was conducted to recognize six DPs in an open source application. The results achieved were comparable with other approaches and in some cases better. Some DPs were almost perfectly recognized and potential improvements have been identified for others.

For the future work, we plan to use more voters (i.e. DP recognition tools) in the training dataset preparation, which will allow us to increase the minimum votes for the positive training examples. We expect this to improve the dataset quality. We also plan to apply rule learning and extraction techniques in order to analyse and compare the extracted rules with those derived from the theoretical descriptions of DPs.

REFERENCES

[1] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language: Towns, Buildings, Constructions*, ser. Center for Environmental Structure Berkeley, Calif: Center for Environmental Structure series. Oxford University Press, 1977.

[2] W. Brown, R. Malveau, H. W. S. McCormick, and T. J. Mowbray, *AntiPatterns: refactoring software, architectures, and projects in crisis*. Wiley, 1998.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994.

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

Table IV
PHASE TWO COMPARSION BASED ON PMART

DP Name	**PMARt**	**Our Approach**		**SSA** [21]		**MARPLE** [18]		**FINDER** [35]	
	Detected	***Detected***	***Shared***	***Detected***	***Shared***	***Detected***	***Shared***	***Detected***	***Shared***
Adapter	**20**	35	2	23	2	98	3	1	0
Decorator	**1**	4	1	3	1	28	1	2	1
Observer	**2**	215	2	3	0	61	1	13	0
Proxy	**0**	0	0	0	0	0	0	0	0
Composite	**5**	12	5	1	1	11	3	21	5
Command	**13**	54	9	23	8	611	13	7	2

[5] F. Buschmann, *Pattern oriented software architecture: a system of patters*. John Wiley & Sons, 1999.

[6] F. Arcelli Fontana and M. Zanoni, "A tool for design pattern detection and software architecture reconstruction," *Information sciences*, vol. 181, no. 7, pp. 1306–1324, 2011.

[7] Z. Balanyi and R. Ferenc, "Mining design patterns from c++ source code," in *Proceedings of The International Conference on Software Maintenance*, 2003, pp. 305–314.

[8] 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. 6, pp. 823–855, 2009.

[9] A. Ampatzoglou, S. Charalampidou, and I. Stamelos, "Research state of the art on GoF design patterns: A mapping study," *Journal of Systems and Software*, vol. 86, no. 7, 2013.

[10] I. Philippow, D. Streitferdt, M. Riebisch, and S. Naumann, "An approach for reverse engineering of design patterns," *Software & Systems Modeling*, vol. 4, no. 1, pp. 55–70, 2005.

[11] Y.-G. Guéhéneuc, J.-Y. Guyomarch, and H. Sahraoui, "Improving design-pattern identification: a new approach and an exploratory study," *Software Quality Journal*, vol. 18, no. 1, pp. 145–174, 2010.

[12] N. Pettersson, W. Lowe, and J. Nivre, "Evaluation of accuracy in design pattern occurrence detection," *IEEE Transactions on Software Engineering*, vol. 36, no. 4, pp. 575–590, 2010.

[13] J.-S. R. Jang, "ANFIS: adaptive-network-based fuzzy inference system," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 23, no. 3, pp. 665–685, 1993.

[14] S. Alhusain, S. Coupland, R. John, and M. Kavanagh, "Towards machine learning based design pattern recognition," in *Proceedings of the 13th Annual UK Workshop on Computational Intelligence*, 2013.

[15] J. Niere, W. Schafer, J. Wadsack, L. Wendehals, and J. Welsh, "Towards pattern-based design recovery," in *Proceedings of The 24rd International Conference on Software Engineering*, 2002, pp. 338–348.

[16] J. Niere, "Fuzzy logic based interactive recovery of software design," in *Proceedings of The 24rd International Conference on Software Engineering*, 2002, pp. 727–728.

[17] D. Dubois and H. Prade, "Possibility theory, probability theory and multiple-valued logics: A clarification," *Annals of mathematics and Artificial Intelligence*, vol. 32, no. 1-4, pp. 35–66, 2001.

[18] M. Zanoni, "Data mining techniques for design pattern detection," Ph.D. dissertation, Università degli Studi di Milano-Bicocca, 2012.

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

[20] S. Uchiyama, H. Washizaki, Y. Fukazawa, and A. Kubo, "Design pattern detection using software metrics and machine learning," in *Proceedings of The First International Workshop on Model-Driven Software Migration*, 2011, pp. 38–47.

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

[22] G. Antoniol, G. Casazza, M. Di Penta, and R. Fiutem, "Object-oriented design patterns recovery," *Journal of Systems and Software*, vol. 59, no. 2, pp. 181–196, 2001.

[23] S. K. Dubey, A. Sharma *et al.*, "Comparison study and review on object-oriented metrics," *Global Journal of Computer Science and Technology*, vol. 12, no. 7, 2012.

[24] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," in *Proceedings of the Fifth International Conference on Dependability of Computer Systems, Monographs of System Dependability*, 2010, pp. 69–81.

[25] M. Dash and H. Liu, "Feature selection for classification," *Intelligent data analysis*, vol. 1, no. 1-4, pp. 131–156, 1997.

[26] I. Kononenko, "Estimating attributes: analysis and extensions of relief," in *Proceedings of The European Conference on Machine Learning*, 1994, pp. 171–182.

[27] A.-H. Tan and H. Pan, "Predictive neural networks for gene expression data analysis," *Neural Networks*, vol. 18, no. 3, pp. 297–306, 2005.

[28] L. Breiman, *Classification and regression trees*. Wadsworth Inc, 1984.

[29] Z. Zhao and H. Liu, "Spectral feature selection for supervised and unsupervised learning," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 1151–1157.

[30] S. L. Chiu, "Fuzzy model identification based on cluster estimation," *Journal of intelligent and Fuzzy systems*, vol. 2, no. 3, pp. 267–278, 1994.

[31] H. He and E. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[32] H. He, Y. Bai, E. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *Proceedings of The IEEE International Joint Conference on Neural Networks*, 2008, pp. 1322–1328.

[33] J. Dong, Y. Zhao, and Y. Sun, "A matrix-based approach to recovering design patterns," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 39, no. 6, pp. 1271–1282, 2009.

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

[35] M. Lebon and V. Tzerpos, "Fine-grained design pattern detection," in *Proceedings of The IEEE 36th International Conference on Computer Software and Applications*, 2012, pp. 267–272.