Towards Machine Learning Based Design Pattern Recognition

Sultan Alhusain and Simon Coupland
Centre for Computational Intelligence
De Montfort University
Leicester, United Kingdom
Email: {sultanib,simonc}@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 are abstract descriptions of best practice solutions for recurring design problems. The information about which design pattern is implemented where in a software design is very helpful and important for software maintenance and evolution. This information is usually lost due to poor, obsolete or lack of documentation, which raises the importance of automatic recognition techniques. However, their vague and abstract nature allows them to be implemented in various ways, which gives them resistance to be automatically and accurately recognized. This paper presents the first recognition approach to be solely based on machine learning methods. We build a training dataset by using several existing recognition tools and we use feature selection methods to select the input feature vectors. Artificial neural networks are then trained to perform the whole recognition process. Our approach is evaluated by conducting an experiment to recognize six design patterns in an open source application.**

Keywords*—*Software design patterns, machine learning, pattern recognition, reverse engineering.

I. Introduction

A software design pattern (DP) can be defined as a recurring structure of classes organized and interact in a particular manner to solve a recurring design problem [1]. In other words, it is a set of communicating classes with a name and a prescribed pair of a design problem to be solved and a solution to that problem. The solution part of DP descriptions specifies best practices in the distribution of responsibilities between participating classes and the collaboration between them [2]. The set of responsibilities assigned to a class defines the role it plays within a DP, and each DP consists of multiple roles played by different classes.

The realization and implementation of a DP in a software design creates what is called a DP instance. It is often that the information about the created instances is lost due to poor, obsolete or even lack of documentation [3] [4]. Consequently, the tasks of software comprehension, maintenance and evolution become difficult, expensive and time consuming [5]. The need for up-to-date documentations posed by these tasks raises the importance of DP recognition techniques as they help to recover valuable software design information.

When DP instances are recognized, a lot of information is recovered and becomes available to maintainers, which enables them to make faster and well-informed maintenance decisions [6]. Examples for the recoverable information include information about roles and responsibilities assigned to some classes, as well as their relationships and interactions with other classes. Also, recognizing a DP instance exposes the design problem which the original designers were trying to solve, which makes the rationale behind the design easy to understand. Because of these great benefits, the research field of DP recognition has been very active recently [7].

In this paper, a DP recognition approach is proposed solely based on machine learning methods. The proposed approach starts with a first phase in which the search space is reduced by identifying a set of candidate classes for each role in every DP. Then, in a second phase, all the possible combinations of related roles' candidates are checked whether they represent a valid instance of a DP or not. A separate artificial neural network (ANN) is trained for each role/DP with a different input feature vector selected by using feature selection methods. The training dataset used for the training is built based on the agreement/disagreement between several existing DP recognition tools. To evaluate our approach, an experiment is conducted to recognize six DPs (i.e. Adapter, Command, Composite, Decorator, Observer and Proxy) in JHotDraw 5.1[1] open source application.

The rest of this paper is organized as follows. Section II presents the motivation for our approach and gives an overview of the problem to be solved. In section III, the two phases of the DP recognition process are discussed in detail. The training dataset preparation steps, including the existing recognition tools used, are presented in section IV. Section V discusses features, feature selection methods, model topology and training. The evaluation results are presented and discussed in Section VI. The related work as well as the conclusions and future work are presented in the last two sections (VII and VIII).

II. Motivation and Overview

There are two interrelated motives and goals behind developing this approach. The first is to fully utilize the power and capability of machine learning methods, and ANNs in particular, in solving the DP recognition problem. Despite the widely acknowledged success of ANNs in solving other pattern recognition problems [8], they have not yet been fully applied in the field of DP recognition. To the best of the authors' knowledge, all what any machine learning method has been

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

used for is to filter out false positives as in [9] and [10], or to reduce the search space as in [11] and [12].

The second motive of developing this approach is to completely learn the recognition rules and features from DP instances implemented in real world applications. Almost all existing DP recognition techniques depend, to different extents, on rules/features derived from theoretical descriptions and are not validated on implemented instances [12]. Some of these techniques are based on the assumption that the theoretical DP descriptions are accurately reflected in the implemented instances although it is not usually the case [13]. To utilize the knowledge hidden in the implemented instances and to improve the recognition accuracy, our approach does not pre-set any rules or features based the DP descriptions. Alternatively, features will be selected by using feature selection methods and rules will be learnt by training ANNs.

Learning rules this way helps to avoid the problem of rules granularity which faces many of the existing recognition techniques [14], assuming that the ANNs are properly trained and not over or under fitted. Setting crisp logical rules is not actually a sensible approach because of the vague and abstract nature of DPs which allows them to be implemented in various ways [15] [16]. Also, since DPs are based on common and conventional object-oriented concepts (e.g. polymorphism), it is likely to have structures of classes that only accidentally have similar relationships as found in some DPs [9]. So, for any DP recognition technique to produce accurate results, it should have the capability to predict the intent behind constructing such structures, and whether or not they are intended to solve the problems for which the corresponding DPs are invented [17]. The failure to do so is likely to result in many false positives. The natural, and probably the best, answer to these problems and challenges lays in the capability and strength of machine learning methods, of which ANN is a powerful technique.

The main drawback of exploiting the full potentials of machine learning methods in DP recognition is the lack of a large standard benchmark by which machine learning models can be trained [18]. In fact, this is a problem even for non machine learning based methods as only a few of them have been independently assessed and validated [19] [20] [21]. To overcome this drawback, a training dataset has to be prepared, which should represent a wide spectrum of application domains with an acceptable level of validity and reliability. The method chosen to build the dataset is based on analysing more than 400 open source applications from various domains by using several DP recognition tools. Then, the recognized DP instances on which the tools agree/disagree are used as positive/negative training examples, respectively. The process of preparing the training dataset is justified and discussed in more details in section IV.

III. The Design Pattern Recognition Process

The pattern recognition problem is solved in two phases as shown in fig. 1. The first phase is dedicated to the recognition of candidate classes which are potentially playing a particular role in a DP. The recognized candidates are then passed to the second phase where the DP recognition is taking place. The two phases are discussed below.

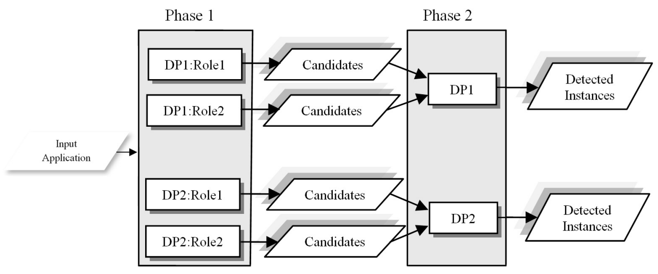


Fig. 1. The overall design pattern recognition process.

A. Phase One (Intra-Class Level)

The primary goal of phase one is to reduce the search space by identifying a set of candidate classes for every role in each DP, or in other words, removing all classes that are definitely not playing a particular role. By doing so, phase one should also improve the accuracy of the overall recognition system. However, these goals or benefits are highly dependent on how effective and accurate it is. Although some false positives are permissible in this phase, its benefits can be compromised if too many candidate classes are passed to phase two (e.g. $\geq$ 50% of the number of classes in the software under analysis). On the other hand, if some true candidate classes are misclassified (they become false negatives), the final recall of the overall recognition system will be affected. So, a reasonable compromise should be struck in phase one and it should favour a high recall at the cost of a low precision.

In the literature, there are two approaches proposing to use machine learning methods in a search space reduction phase. In the first approach [12] [13], the same set of 13 metrics was calculated for all classes playing all roles. Then, a rule learner was used to infer a set of rules for each individual role, which can then be used to identify candidate classes for them. Our approach, on the other hand, use an extended set of class level features, and then use feature selection methods to select a different subset of features for each individual role. Also, while their approach used a small repository of DP instances taken from only 9 open source systems (PMARt [20] [22]), our approach use a large training dataset prepared from more than 400 open source applications representing a wide range of application domains.

In the second approach, a single ANN was trained to identify candidates for all roles in all DPs [11]. However, because a single class can play multiple roles in different DPs, the same set of input feature values for a class can sometimes have different target outputs. This problem is called multi-label classification problem [23] and it was apparently ignored in their approach. This problem is addressed in our approach by having multiple single-label ANNs, one for each role. This is actually beneficial and desirable as it allows for the selection of a different feature set to best suit each individual role.

Although DPs consist of multiple roles, only key and characteristically unique roles are considered in our approach. The set of roles recognized is called “the occurrence type” and their number is called the “occurrence size” [18]. Different approaches use different occurrence types and sizes, and our approach adopts the same occurrence as the one used in [24] for the same reason. In [24], it is suggested that the inclusion of non-key roles, e.g. roles that only participate in inheritance relationships with other roles, would result in too many false positives. Some of the non-key roles can actually be missing without affecting the existence of their DPs, and a number of examples for such cases are given in [3]. The decision to only consider key roles is also supported by the experiments reported in [12] and [13], in which non-key roles were described as not having a fingerprint or a numerical signature. Moreover, it was found in [18] that the precision of DP recognition decreases when more roles are considered. Nevertheless, after recognizing the key roles of a DP, it will be easy to manually identify non-key roles as they are usually closely related. Table I shows the roles/DPs considered in this paper.

B. Phase Two (Inter-Class Level)

In this phase, the core task of DP recognition is performed by examining all possible combinations of related roles' candidates. Each DP is recognized by a separate classifier, which takes as input a feature vector representing the relationships between a pair of related candidate classes. Similarly to roles in phase one, different DPs have different subsets of features selected to best represent each one of them. Input feature vectors and model training are discussed in section V.

Without being preceded by a search space reduction phase, phase two can suffer from the combinatorial explosion problem because it will exhaustively check all possible pairs of classes in the software under analysis. Let us say, for example, that there are n and m candidate classes for role1 and role2, respectively. Let us also say that the total number of classes in a software system is T. With a space reduction phase, the number of the possible combinations between the two roles' candidates will be $n \times m$, where $n, m < T$. Otherwise, the possible combinations to be checked will be $T \times (T-1)$, which can be far more computationally expensive.

IV. Training Dataset Preparation

The ideal method of preparing the training dataset is to manually analyse open source applications by a team of experts in order to compile a comprehensive list of peer reviewed DP instances. However, this method will be prohibitively time consuming as it requires careful study and deep understanding of each analysed application, some of which can have obsolete, poor or no documentation at all. This ideal method is suggested to be beyond the ability of a single research group [18].

The next best alternative, and probably the best possible, is to automatically prepare the training dataset by using several existing DP recognition tools. This can be done by running the recognition tools on a set of open source applications and calculate how many votes each recognized instance receives (i.e. how many tools recognize each instance). We assume that the more tools a DP instance is detected by, the more likely it is to be a true positive instance. This assumption is based on the fact that different recognition tools are based on different recognition approaches which use varying sets of characteristics and recognition rules. So, when an instance is reached from different approaches, it has, at least theoretically, a much better chance to be a true positive.

A. Voters

Four DP recognition tools are used as voters in the training dataset preparation. They are selected simply because of their capability to export the recognition results into XML based files, which makes it much easier for the recognition results to be automatically manipulated. The tools selected are as follows:

- SSA (Similarity Scoring Algorithm) [24].
- MARPLE (Metrics and Architecture Reconstruction PLugin for Eclipse) [6] [10].
- WOP (Web of Patterns) [25] [26].
- FINDER (FINe-grained DEtection Rules) [27].

Besides these tools, a semi-automatically prepared repository (Percerons [28] [29]) is also used as a voter. DP instances were added to Percerons repository either when they were detected by two tools or only by one with an approval from an expert. Although SSA was one of the two tools used to build Percerons, there is no duplication in votes. This is because each DP instance in Percerons is also detected by the other tool (PINOT [30]) or manually approved by an expert.

B. Open Source Applications

To make all the voters vote on the same set of DP instances, the same applications included in the Percerons repository are also analysed by the four recognition tools. The total number of the analysed open source applications is 433, which represent a wide spectrum of application domains. The list of domains covered include for example: science and engineering, audio and video, business enterprise, communication and game application domains. This is important as it contributes to the validity of any knowledge derived from the dataset.

C. Voting System

A chain of java programs are developed to convert the XML files produced by each of the recognition tools into a common format, and then to conduct the votes calculation. Fig. 2 shows the training data preparation and vote calculation steps. Beside the vote summary table which is generated at the end of these steps, multiple folders are also created to store instances based on the number of votes they received. Due to the space limitation, the vote summary table is not included in this paper.

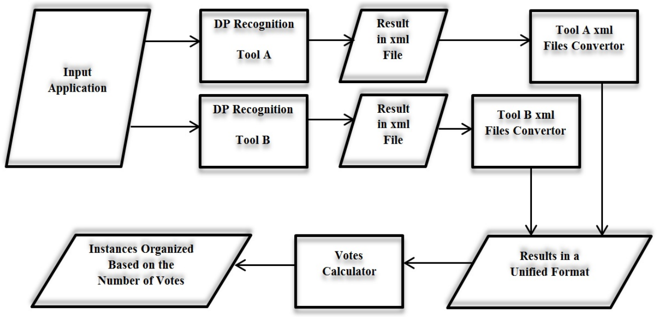

Input Application
DP Recognition Tool A
Result in xml File
Tool A xml Files Convertor
DP Recognition Tool B
Result in xml File
Tool B xml Files Convertor
Instances Organized Based on the Number of Votes
Votes Calculator
Results in a Unified Format

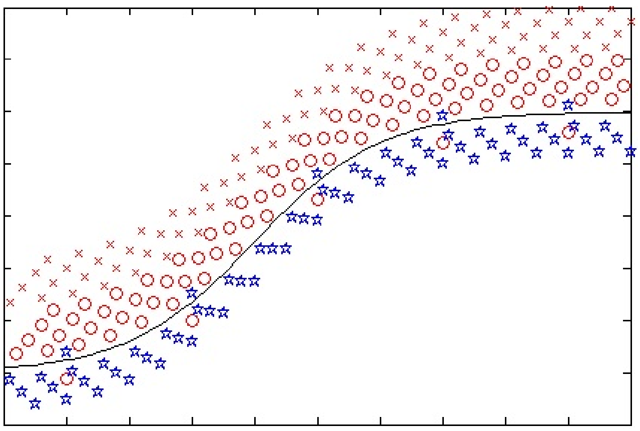
Fig. 2. Training dataset preparation and vote calculation.

D. Positive and Negative Training Examples

Positive and negative training examples are needed for each role/DP to train their classifiers. The positive training examples can include all instances with a minimum number of votes. This minimum is set to two in the experiment reported in this paper, which means that each positive training example in our dataset has been voted for by at least two recognition tools. However, for the Adapter and Command DPs and their roles, the minimum is increased to at least three to overcome the confusion caused by the inability of two voters to distinguish these two DPs.

There are two options with regard to the negative training examples. The first one is to use instances with zero votes as negative training examples. However, there would probably be too many of them, as they will simply be every class and pair of classes which have not been voted for by any of the voters. The second option, which is the one adopted, is to use instances with one and exactly one vote. Since there are several voters, it can be safely assumed that instances with a single vote are mostly false positives. We cannot obviously guarantee that no false positives will receive more than a single vote, which may result in incorrectly labelling some of them as true positive examples. However, such mislabelled training examples can just be considered as 'noisy' examples, which are normally found in many training datasets.

Adopting the second option also can improve the recognition performance. Instances with two votes are more likely to be characteristically closer to those with one vote than to those with zero votes. This is based on the fact that instances with one vote should at least exhibit some characteristics that trick a tool into thinking that they are valid instances of a DP while they are not. It is these characteristics that make instances with one vote (i.e. negative examples) closer to those with two or more votes (i.e. positive examples). So, drawing the decision boundary between the positive examples and their closest negative examples will probably result in a DP recognition system with a high accuracy. This idea is illustrated in Fig. 3.



Instances in the universe of discourse: the stars represent instances with two or more votes, the circles represent instances with exactly one vote, and the crosses represent instances with zero votes.

Fig. 3. An ideal example of how classification decision boundary should look like.

V. Feature Selection and Model Training

A. Features and Feature Selection

There is still a lack of research on the characteristics/features which should be checked to recognize DPs [3]. Almost all the features used so far are limited to those based on or derived from the theoretical description of DPs. The only exception is [12] in which the external quality attributes of classes (e.g. coupling) are used to reduce the search space, as discussed earlier. So, to have representative feature sets that are not limited to the theoretical descriptions, an extended set of features is compiled from which a different subset is selected for each role/DP.

The extended feature set includes most of the features used in [11], [12], [17] and [24]. It also includes many object oriented metrics like RFC (Response for class) [31] and Ca (Afferent couplings) [32]. The total number of features included is 82 metrics for phase one and 112 metrics for phase two. All the 82 features in phase one are class-level metrics capturing different aspects and properties of individual classes. Although phase two is an inter-class phase, 88 features out of the 112 are class-level metrics, which are again included in this phase to capture any valuable information hidden in the relativity between intra and inter class features. Such information was successfully used to filter out false positives in [9] (e.g. the ratio between the number of inherited public methods in a class which invokes an external method and the total number of public methods in the class). The rest of the features included in phase two represent the relationships and interactions between classes.

All the features are calculated by using CKJM[2], JMT[3], POM[4] and Dependency Finder[5] tools. Also, the underlying tool developed by [24] is modified and used. The modification is performed so instead of, for example, extracting Boolean values representing the existence of a method invocation between two classes, it counts how many methods of a class contains a method invocation to another class. This change is expected to provide valuable information to the classifiers.

Different roles/DPs probably have different best subset of features. The best subset for each roles/DPs is the one which has values very similar for instances of the same role/DP and very different otherwise. It is also the one in which "the true (unknown) model of the patterns can be expressed" [33]. Feature selection methods can be used to find the best feature subsets. However, searching for the best optimal subset can be computationally expensive and prohibitive. So, many feature selection methods compromise optimality for performance by using search methods that find good feature subsets but not necessarily the best ones [34].

A widely used category of feature selection methods is the one in which individual features are evaluated and ranked [35]. In our approach, multiple feature selection methods are used and all of them fall under this category. The methods used are reliefF [36], fisher [37], Gini index [38] and spectrum [39]. All of these methods are used to generate multiple differently

[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

TABLE I. PHASE ONE EVALUATION RESULT

DPName:RoleName	Threshold = 0					Threshold = $-\frac{1}{3}$				
	TP	*FP*	*FN*	*Precision*	*Recall*	*TP*	*FP*	*FN*	*Precision*	*Recall*
Adapter:Adaptee	6	18	5	25	54.6	11	41	0	21.2	100
Adapter:Adapter	15	17	11	46.9	57.7	24	46	2	34.3	92.3
Decorator:Component	3	0	0	100	100	3	0	0	100	100
Decorator:Decorator	4	4	0	50	100	4	6	0	40	100
Observer:Subject	3	55	0	5.2	100	3	69	0	4.2	100
Observer:Observer	2	22	0	8.3	100	2	44	0	4.4	100
Proxy:RealSubject	0	0	0	100	100	0	11	0	0	100
Proxy:Proxy	0	0	0	100	100	0	0	0	100	100
Composite:Component	1	7	0	12.5	100	1	28	0	3.5	100
Composite:Composite	2	9	3	18.2	40	5	22	0	18.5	100
Command:Receiver	4	17	0	19.1	100	4	33	0	10.8	100
Command:ConcreteCommand	7	23	6	23.3	53.9	13	48	0	21.3	100

ordered feature sets for each role/DP. Then, the classifiers of all roles/DPs are trained and tested with all of their feature sets by using various numbers of features. The feature subsets that yield the best performances are the ones finally selected.

It is noteworthy that most of the features, which are known to be important, have been highly ranked by the selection methods. For example, the number of inherited methods in the adapter class with a method invocation to the adaptee class is an important feature that characterises instances of the Adapter DP. The fact that such features are recognized as important and discriminating features supports our assumption that instances with only one vote are highly likely to be false positives, and hence they are used as negative training examples. If the assumption was not true at all, these important features would probably have not been given high weights because they would have similar ranges of values regardless of the number of votes.

B. Model Topology and Training

The prepared training dataset has far more negative training examples than positive ones. The problem of having such an imbalanced dataset can significantly affect the performance of most learning algorithms. This is a common problem encountered in many real world applications and several approaches have been proposed to deal with it [40]. One possible approach is based on the use of random over and under-sampling. However, this may result in a loss of important information in the case of under-sampling and over-fitting in the case of over-sampling. Informed synthetic sampling approaches (like ADASYN [41]) is probably a better alternative. ADASYN is a synthetic sampling technique based on identifying the minority seed examples and then synthesizing extra neighbouring samples for them. The ADASYN technique is used in the experiment reported in this paper.

The experiment is conducted by using feed-forward ANN with back propagation training algorithm. In ANNs, finding the best model topology, e.g. the number of hidden layers and nodes, is essential and crucial to achieve a good performance. Too many hidden nodes will result in too complex decision boundaries which lead to low training error but poor generalization performance (i.e. over-fitting). On the other hand, too few hidden nodes will result in high training and generalization errors (i.e. under-fitting) [33]. The best topology depends on many factors including the number of input features and training examples as well as the complexity of the classification problem. However, there is no standard way of determining the best topology and it traditionally depends on trial and error [42]. So, the ANNs of all roles/DPs are trained multiple times by using different topologies and input feature vectors. Each of these different configurations is evaluated based on the 10-fold cross validation, and the topology selected for each role/DP is the one that gives the best performance (lowest average mean squared error).

VI. EVALUATION RESULTS

The absence of a standard benchmark makes it hard to objectively evaluate and validate the accuracy of DP recognition approaches [43]. Most approaches in the literature are evaluated by manually checking results generated. However, such evaluation is subjective since the same instance can be labelled as a false positive by one evaluator and as a true positive by another. So, in order to have an objective evaluation, no instance in the recognition test results is labelled based on our own judgment. Alternatively, the only available peer reviewed repository (PMARt [20]) is used as a main reference for the evaluation. Since this repository is not claimed to be complete, the internal documentation of the application under analysis (JHotDraw 5.1) as well as the relevant literature are also considered during the evaluation.

Because JHotDraw is designed based on some well-known DPs, it has become a popular choice for evaluating many DP recognition approaches. The design of JHotDraw has been manually examined and the identified DP instances have been peer reviewed and added to the PMARt repository. To ensure the validity of our evaluation, JHotDraw is not added to the list of applications from which the training dataset is prepared.

The proposed approach is implemented in MATLAB and it is evaluated by applying it to recognize DP instances in

TABLE II. PHASE TWO EVALUATION RESULT

DP Name	Threshold = 0				
	TP	*FP*	*FN*	*Precision*	*Recall*
Adapter	6	6	19	50	24
Decorator	4	1	0	80	100
Observer	3	268	0	1.1	100
Proxy	0	0	0	100	100.0
Composite	4	26	1	13.3	80
Command	8	33	5	19.5	61.5

TABLE III. Phase One Comparsion Based on PMARt

DP Name	PMARt	Our Approach		SSA		MARPLE		FINDER	
	Detected	*Detected*	*Shared*	*Detected*	*Shared*	*Detected*	*Shared*	*Detected*	*Shared*
Adapter:Adaptee	**8**	52	8	13	3	30	6	1	0
Adapter:Adapter	**21**	70	19	21	3	46	5	1	0
Decorator:Component	**1**	3	1	3	1	14	1	1	1
Decorator:Decorator	**1**	10	1	3	1	21	1	2	1
Observer:Subject	**2**	72	2	2	0	38	2	10	0
Observer:Observer	**2**	46	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**	29	1	1	1	5	1	4	1
Composite:Composite	**5**	27	5	1	1	9	3	9	5
Command:Receiver	**4**	37	4	13	3	116	4	2	2
Command:ConcreteCommand	**13**	61	13	21	8	111	13	7	5

JHotDraw application. The recognized DP instances are then examined individually and labelled as true or false positive. Precision and recall, which are common information retrieval accuracy measures, are calculated for the ANNs of all roles/DPs. The precision shows how much of the retrieved (recognized) instances are relevant (i.e. true positives). The recall shows how much of the relevant instances (i.e. all existing instances) are retrieved. Similarly to some other recognition approaches, DP instances with common classes are counted as different instances and not aggregated into one instance.

Table I shows the number of true positives, false positives and false negatives as well as the precision and recall values for each role. Misclassifying true instances in phase one will negatively affect the recall of the overall system while the opposite has less effect, as discussed earlier. So, the threshold is adjusted to ensure that as many true positive candidates as reasonably possible are passed into phase two. Threshold-moving is actually one of the recommended techniques to deal with classification problems with different misclassification costs [44]. The results obtained with 0 as well as $-\frac{1}{3}$ thresholds are shown in table I. It can be seen that even after moving the threshold to $-\frac{1}{3}$, the search space is still reduced by around 79% on average (JHotDraw 5.1 has 173 classes). In the same time, a perfect recall is achieved for all roles apart from the adapter role which has a recall of 92.31%. So, since phase one is mainly a search space reduction phase, the results achieved in this phase can be considered satisfactory.

The results of phase two are presented in table II. Although a low precision is achieved (43.99% on average), it is comparable to the precision achieved by some other approaches like DeMIMA, which has an average precision of 34%. A relatively better recall is achieved at an average of almost 70%. The desired trade off between high precision and high recall depends on the objective of the recognition technique. For the objective of program comprehension, higher recall is considered to be more important [4].

As can be seen in table II, there are too many false positives for the Observer DP, which negatively affects the precision average. However, the use of another over-sampling technique has already shown a significant improvement in the recognition of this DP, and so this will be investigated in our future work. The low precision and recall rates of the Adapter and Command DPs can be partially attributed to the confusion observed between their results. The confusion is probably caused by the inability of some voters to accurately distinguishing them from one another. Using more voters and increasing the minimum vote required for positive training examples in training dataset preparation is a possible solution for such confusion.

Although PMARt is not a complete repository, it is still the only available peer reviewed one. So, comparing different recognition approaches, based on how many instances they recognize/share with it, can represent a useful indicator for accuracy. Since PMARt is not complete, instances which are not shared with it are not necessarily false positives but only likely to be so.

Table III presents a comparison between our approach and three other approaches at the level of the recognized roles. The table shows that our approach has a total of 56 shared instances out of 58 instances included in PMARt while the other approaches have at most 38 shared instances in total. Although our approach has relatively more unshared instances than two of the other approaches, it is not the final result of our approach but only the result of a space reduction phase.

The comparison between the approaches at the level of the

TABLE IV. Phase Two Comparsion Based on PMARt

DP Name	PMARt	Our Approach		SSA		MARPLE		FINDER	
	Detected	*Detected*	*Shared*	*Detected*	*Shared*	*Detected*	*Shared*	*Detected*	*Shared*
Adapter	**20**	12	1	23	2	98	3	1	0
Decorator	**1**	5	1	3	1	28	1	2	1
Observer	**2**	271	2	3	0	61	1	13	0
Proxy	**0**	0	0	0	0	0	0	0	0
Composite	**5**	30	4	1	1	11	3	21	5
Command	**13**	41	8	23	8	611	13	7	2

recognized DPs is presented in table IV. In total, our approach shares more instances with PMARt than SSA and FINDER at the cost of having more unshared instances. Similarly, MARPLE shares 5 more instances than our approach but with 450 more unshared instances.

Overall, it can be seen that the experimental results are encouraging and comparable to the other approaches. So, we can say that the ANN, as a machine learning method, represent a viable and promising approach for DP recognition.

VII. Related Work

Many DP recognition approaches have been proposed but only a few has attempted to make use of the advantages of machine learning or artificial intelligence (AI) techniques in general. In this section, only the recognition approaches which use AI related techniques are presented and briefly discussed.

In [14] [45], a DPs detection approach was proposed based on graph theories. In this technique, patterns were defined in terms of sub-patterns, which are common sub-structures between DPs. Then, the problem of identifying patterns in a software system was handled as a sub-graph isomorphism problem. To solve this sub-graph isomorphism problem, patterns (and sub-patterns) were formally defined as graph transformation rules. To avoid the problem of defining a rule for each possible variant to be recognized, similar rules were replaced by an abstract one, which introduced a level of uncertainty. So, 'fuzzy beliefs' were assigned to each rule to express the degree of this uncertainty. The fuzzy beliefs can simply be calculated for each rule by dividing the number of the correct matches by the number of all matches. Despite being clearly based on probability, the technique was incorrectly claimed to be a "fuzzy logic based" [46].

Another graph based technique was proposed in [15]. This recognition technique was performed in two main phases. First, a set of candidate classes were identified for each role in every DP. Like our approach, metrics were used to identify a set of candidate classes. However, the metrics were selected based on the theoretical description of DPs and they were used to set crisp logical rules, which is likely to result in many false positives and negatives. In the second phase, a detailed analysis is performed on all combination of candidate classes to find DP matches. To improve the result precision, machine learning techniques (decision tree and ANNs) were used to filter out as many false positives as possible and to distinguish similar patterns [9].

Another technique that also use graph matching to recognize DPs was introduced in [6] and [10]. In this recognition technique, DPs were modelled based on microstructures, which are facts or relationships between code entities. The system to be analysed was represented as a graph in which edges represent the microstructures and nodes represent the classes. Then, graph matching techniques were applied on the system graph to identify DP candidates. All the identified candidates were then analysed by a classifier module to filter out false positives.

Two more approaches proposed in the literature ([13] and [11]) in which learning models were used to reduce the search space. These two approaches have already been discussed in comparison with our approach in section III.

VIII. Conclusions and Future Work

We have presented a DP recognition approach based on ANNs and feature selection techniques. Our approach does not use any pre-set rules or features derived from the theoretical descriptions of DPs. Alternatively, recognition rules and features are derived from the knowledge hidden in instances implemented in real world applications. The training dataset, from which the knowledge is derived, is prepared by using several existing DP recognition tools based on the agreement/disagreement between their results.

The proposed approach was evaluated by conducting an experiment to recognize six DPs in an open source application. The results achieved are encouraging and comparable to other existing approaches, which show that our approach is viable and promising. This strengthens our belief that ANNs, as a powerful pattern recognition technique, can be successfully used to solve the problem and challenges of software DP recognition.

For the future work, we plan to use more voters (i.e. DP recognition tools) in the training dataset preparation, which will enable us to increase the minimum votes required for positive training examples. Increasing this minimum will improve the training dataset accuracy, which will probably improve the performance of classifiers. We also plan to use other artificial intelligence learning methods, such as support vector machine (SVN) and adaptive network based fuzzy inference system (ANFIS). Our plan also includes the application of rule learning and extracting techniques, and compare the extracted rules with those derived from the theoretical descriptions.

References

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

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

[3] 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.

[4] Y.-G. Gueheneuc and G. Antoniol, "DeMIMA: A multilayered approach for design pattern identification," *IEEE Transactions on Software Engineering*, vol. 34, no. 5, pp. 667–684, 2008.

[5] 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.

[6] F. Arcelli and L. Cristina, "Enhancing software evolution through design pattern detection," in *Proceedings of The Third International IEEE Workshop on Software Evolvability*, 2007, pp. 7–14.

[7] 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.

[8] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] J. Smith and D. Stotts, "Spqr: flexible automated design pattern extraction from source code," in *Proceedings of The 18th IEEE International Conference on Automated Software Engineering*, 2003, pp. 215–224.

[17] 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.

[18] 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.

[19] F. Arcelli, M. Zanoni, and A. Caracciolo, "A benchmark platform for design pattern detection," in *Proceedings of The 2nd International Conference on Pervasive Patterns and Applications (PATTERNS)*, 2010.

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

[21] L. Fulop, R. Ferenc, and T. Gyimothy, "Towards a benchmark for evaluating design pattern miner tools," in *Proceedings of The 12th European Conference on Software Maintenance and Reengineering*, 2008, pp. 143–152.

[22] Y.-G. Guéhéneuc. P-mart: Pattern-like micro-architecture repository. [Online]. Available: http://www.ptidej.net/tool/designpatterns/

[23] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 3, no. 3, pp. 1–13, 2007.

[24] 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.

[25] J. Dietrich and C. Elgar, "A formal description of design patterns using owl," in *Proceedings of The Australian Software Engineering Conference*, 2005, pp. 243–250.

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

[27] Finder [design pattern detection]. [Online]. Available: https://wiki.cse.yorku.ca/project/dpd/finder

[28] A. Ampatzoglou, O. Michou, and I. Stamelos, "Building and mining a repository of design pattern instances: Practical and research benefits," *Entertainment Computing*, 2012.

[29] Percerons: Easy retrival and adaption of open source software components. [Online]. Available: http://www.percerons.com/cbse.html

[30] N. Shi and R. Olsson, "Reverse engineering of design patterns from java source code," in *Proceedings of The 21st IEEE/ACM International Conference on Automated Software Engineering*, 2006, pp. 123–134.

[31] 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.

[32] 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.

[33] R. Duda, P. Hart, and D. Stork, *Pattern classification*. Wiley, 2001.

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

[35] A. Arauzo-Azofra, J. L. Aznarte, and J. M. Benítez, "Empirical study of feature selection methods based on individual feature evaluation for classification problems," *Expert Systems with Applications*, vol. 38, no. 7, pp. 8170–8177, 2011.

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

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

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

[39] 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.

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

[41] 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.

[42] D. Stathakis, "How many hidden layers and nodes?" *International Journal of Remote Sensing*, vol. 30, no. 8, pp. 2133–2147, 2009.

[43] 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.

[44] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 63–77, 2006.

[45] J. Niere, J. Wadsack, and L. Wendehals, "Handling large search space in pattern-based reverse engineering," in *Proceedings of The 11th IEEE International Workshop on Program Comprehension*, 2003, pp. 274–279.

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