Artificial Intelligence Review (2020) 53:5789–5846
https://doi.org/10.1007/s10462-020-09834-5

Design pattern detection approaches: a systematic review of the literature

Hadis Yarahmadi[1] · **Seyed Mohammad Hossein Hasheminejad**[1]

Published online: 20 April 2020

Abstract
Nowadays, software engineers have a great tendency to use design patterns (DPs) because they are considered to have an important role in software engineering in the sense that they can make the understanding of nonentities easier. However, most of the systems have no document helping engineers recognize DPs from the codes. As a result, different approaches for design pattern detection have been suggested. The current paper reviews different available literature on design pattern detection and reports a number of different aspects of them such as data representation, type of design pattern, advantages and disadvantages for different approaches, quantitative results, etc. The current paper reviews research studies published between 2008 until 2019 and represents a list of datasets used for evaluations. The present investigation paper is not only to lay the ground for the selection of the optimal design patterns, but also hopes to guide the future studies through raising awareness about the potential defects in the previous researches.

Keywords Design pattern detection · GoF · Object-oriented source codes · Systematic review

1 Introduction

Design patterns (DPs) are commonly used in software design phases to solve most of the problems in engineering and to facilitate the software development process. Nowadays, design patterns are used in the design phase, but their use is not trouble-free because DPs are considered in the design phase and sometimes they are forgotten in the implementation. According to Gamma (1995), design patterns can be classified into three categories including structural, behavioral and creational. The design patterns apply experts' knowledge to solve common software problems which have similar solutions (Riaz et al. 2015). Generally, design patterns are very important because they are helpful in reverse engineering

✉ Seyed Mohammad Hossein Hasheminejad
SMH.Hasheminejad@Alzahra.ac.ir

Hadis Yarahmadi
H.Yarahmadi@Alzahra.ac.ir

[1] Department of Computer Engineering, Alzahra University, Tehran, Iran

and system understanding (Kaczor et al. 2010). Design patterns should be used in software development mainly because:

- They increase the speed of software development.
- They prevent unforeseen problems.
- They can help the programmers to easily understand the code and solve the problem.

Design patterns enhance reusability of modularization, the quality and consistency of design and implementation, and the relationship between design and implementation teams. They also make the system flexible and scalable (Thongrak and Vatanawood 2014). They are also very important when a developer tries to regenerate a previously generated code for a particular system (Thankappan and Patil 2015). In fact, one of the challenges of design pattern use corresponds to code and design. Moreover, the variant implementation of a design pattern makes it difficult to be detected from source code. Design pattern detection, using whatsoever approach, can help us when the documents are not available. Design pattern detection (DPD) is important for several reasons. First, design pattern detection provides design information needed for software architecture rebuilding. In other words, design patterns make it possible to compare code and to check if the source code and the design correspond to each other (Fontana and Zanoni 2011; Wierda et al. 2008; Tsantalis et al. 2006). Table 1 represents the questions for this research paper. In the field of DPD approaches, some secondary studies have reviewed the literature. It should be noted that none of the reviews on design pattern detection meets researcher's needs because they are limited in their scope and have not reviewed new papers. The categorization suggested in (Al-Obeidallah et al. 2016) is too broad in the sense that it does not pay attention to the specificities of the learning-based approach despite its capacity to bring about a drastic change in design pattern detection. As another example, Priya (2014) and Dong et al. (2009a) only consider a single perspective. However, the scope of our

Table 1 The questions for research

RQ1: What is the design pattern detection?
RQ2: What kinds of design patterns are used in different publications?
RQ3: What items are considered as inputs and which ones are most frequently used?
RQ4: What kinds of the input representations are used in the papers and which ones have been most used?
RQ5: What is the difference between some kinds of Data Representation such as ASG, AST, Feature vector and metrics?
RQ6: What basis the Different methods choose their input representation method?
RQ7: What types of approaches are there in design pattern detection and which one of them have been most frequently used?
RQ8: What are the advantages of each approach?
RQ9: What are the disadvantages of each approach?
RQ10: What are the datasets that are used in different methods for evaluation and which ones have been most frequently used?
RQ11: What are the conclusion and comparison of the quantitative results of different approaches?
RQ12: which approaches have high precision and recall? What are our suggestions in using one of these techniques?
RQ13: What are the open issues for the future and what can be done to improve performance

study is all the DPD approaches that exist in the literature, while previous reviews have limited perspective of DPD approaches. Moreover, our paper covers about more years, and a larger number of publications and venues. These differences make this work substantially different and worth investigation. In the related Works section, the comparison between our work and other secondary studies in the field has been conducted in detail. To compensate for the deficiencies of previous reviews, the current paper subsumes the recently published research, i.e., those published during 2008–2017, and opts for a micro view through which different approaches are carefully examined and their advantages and disadvantages are compared. Al-Obeidallah et al. (2016), Priya (2014) almost neglected the input and its pre-representation. Al-Obeidallah et al. (2016) cursorily do not touch the different datasets. It also did not say anything about the different popularity levels of the design patterns for detection. However, this paper examines dataset types and schematizes them. Although none of the previous studies yielded quantitative results of the functionality of different approaches with various datasets, the current study reported the results quantitatively, as well. These quantitative results provide researchers with information including the detection accuracy of different patterns and the frequency of their use so that they can decide what patterns should be studied in future. Moreover Al-Obeidallah et al. (2016), Priya (2014), Dong et al. (2009a) only familiarize researchers with some basic general concepts of design pattern detection and fail to provide any clear conclusions. By contrast, this study does not beat around the bush and clearly shows the open issues, which have not been studied adequately and are yet to probe. This paper reviews the recent literature on design pattern detection in details, provides clear insights of the concepts, offers comprehensible quantitative results, and as a result, directs the future researches. The rest of the paper organized as follows: related work is presented in Sect. 2. In Sect. 3, we illustrate how to select and extract information from articles. In Sect. 4, different kinds of design patterns which are used in different publications are reviewed. In Sects. 5, 6, different kinds of inputs and their representation are reviewed. Section 7 introduces approaches used in Design Pattern Detection (DPD) and the literature is categorized according to techniques. In Sect. 8, the advantages and disadvantages of different approaches have been checked in different aspects and we have also tried to provide a comprehensive overview of the approaches, which can be helpful for researchers interested in this field. In Sect. 9, datasets that have been used in research papers are reviewed and quantitative result of the recent paper and most are illustrated in Sect. 10. In Sect. 11, open issues will be considered in this context and in Sect. 12, the conclusion of paper will be presented. Finally, we will comprehensively discuss the researches and their results in "Appendix".

2 Related works

Searching on the DPD field, we found secondary studies that use a systematic method to review the literature. Secondary study replications are valuable because, whenever new evidence is published after the completion of a secondary study, we should update it. Among these studies, the only work with the same goals is the work of Al-Obeidallah et al. (2016). The context of our study is all the DPD approaches that exist in the literature, while Al-Obeidallah et al. (2016), only considered limited number of approaches.

This paper compared to other existing works, covers a larger number of publications and venues. This study contributes into providing a taxonomy of the research topics related to DPD approaches as well as a breakdown of papers in each topic. Moreover, the researchers, countries, and publication venues that are active in research related to DPD approaches are identified. Determining the taxonomy and demographics of a particular research topic can provide a useful starting point for other researchers. Therefore, using results that are extracted from this study, we want to present a reference guide for those researchers who seek to enter the DPD approaches field and need to identify research trends and gaps.

In this paper, we present complete results and analysis to those researchers who seek to enter the field of DPs. Hence, as mentioned before, the result of our work can be used as a reference guide and we have specifically discussed what can be used in future works.

Figure 1 represents range of activity per each year. In 2010, most researches have conducted a few studies in design pattern detection. The geographical distribution of authors and those who are most active are shown in Figs. 2 and 3, respectively. As shown in Fig. 2, Italy has the most activity in this field and as shown in Fig. 3, Manjari Gupta has been the most active researcher in this field.

3 Research method

This research was conducted according to the procedure used by Kitchenham and Charters (2007), which, in turn, fits into three phases. The first phase involves planning the review and creating the research protocol to be followed. The second phase entails conducting the review, subsequently performing the search, selecting the papers, and extracting and synthesizing data. The third phase takes reporting the review and specifying the dissemination mechanisms and formats of the main report. In this sense, such a phase resulted in the elaboration of this paper. In this section, we describe how the first two phases were executed.

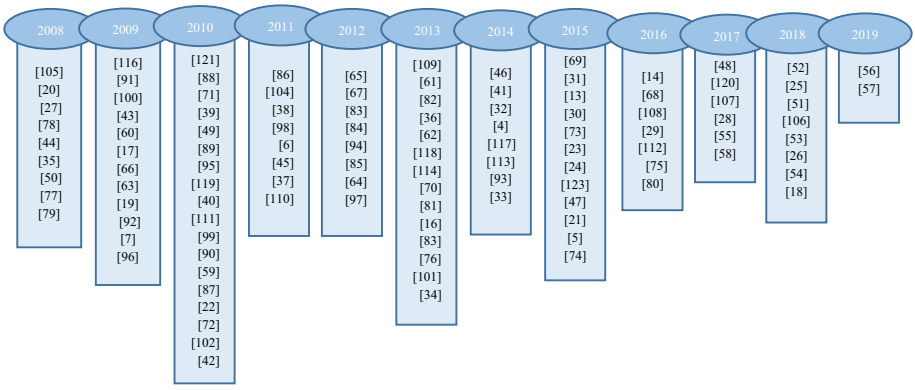

Fig. 1 Number of DPD papers published per year

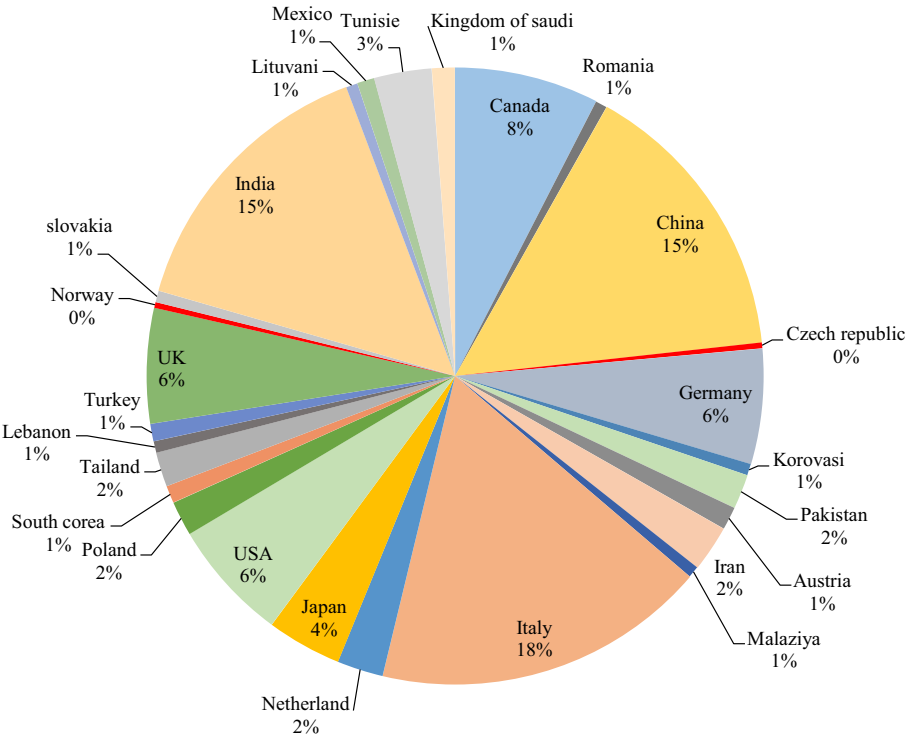

Fig. 2 Geographical destribution of the authors of the papers in DPD (including duplicates)

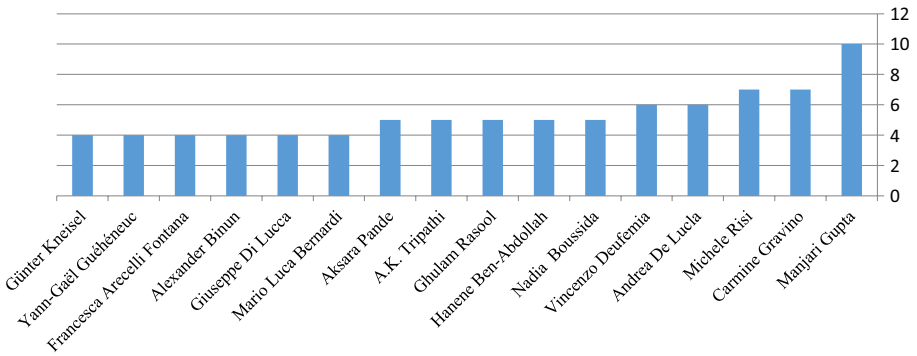

Fig. 3 Authors with the greater number of publications in DPD

3.1 Planning the review

The main goal of the review is to provide an overview of this field and to analyze the approaches' elements, such as design pattern types, search methods, input representations, evaluation methods and results.

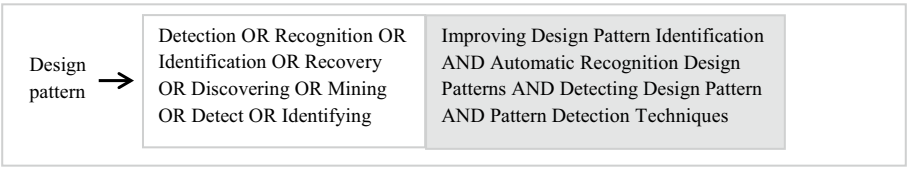

Fig. 4 Example of search strings

Table 2 Inclusion and exclusion criteria

Inclusion	Exclusion
Papers that use any technique to identify/select a component from an artifact is given as input	Out of scope; not available online; Not in English or Persian; abstracts, posters, technical reports, thesis, keynote, doctoral symposiums, book reviews, and patents

3.2 Research question

Our review aims at answering the research questions presented in Table 1. They were defined with the goal of covering the main aspects of approaches. In the following sections, each question in Table 1 will be answered.

3.3 Search string

To build the search string (Fig. 4), we defined the main terms and synonymous terms identified by analyzing search strings used in related works.

3.4 Paper selection criteria

We established a set of inclusion and exclusion criteria adopted in selecting the primary sources presented in Table 2. First, the papers were chosen according to the inclusion criteria. Then, the exclusion criteria were applied in the selected papers. A paper was included or excluded by reading it in the following order: title, abstract, introduction, conclusion, and the entire paper; by doing so, there is no doubt in the selection.

3.5 Paper selection criteria

The first step was the selection of primary sources. This can be fulfilled by using the search string in the selected engines; needless to say, the title, abstract, and keywords should be considered in this respect. The engines for making the search are online repositories chosen based on the fact that they win popularity and they provide many leading software engineering publications. The selected sources are as follows: (1) ACM Digital Library, (2) IEEE Xplore, (3) Springer, (4) Science Direct, (5) Web of Science, and (6) Wiley Online Library. Table 3 illustrates the resource for extracting papers. As shown in Fig. 5, IEEE

Table 3 Resource reviewed

Source	URL
SCOPUS	https://www.scopus.com
IEEE	http://ieeexplore.ieee.org
Springer	https://www.springerlink.com
ACM	https://dl.acm.org
Science Direct	http://www.sciencedirect.com
Wiley Online Library	http://onlinelibrary.wiley.com
Other	–

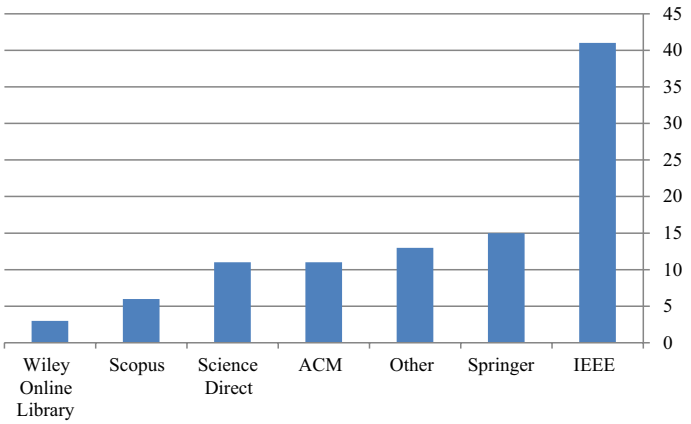

Fig. 5 Distribution of papers in resources

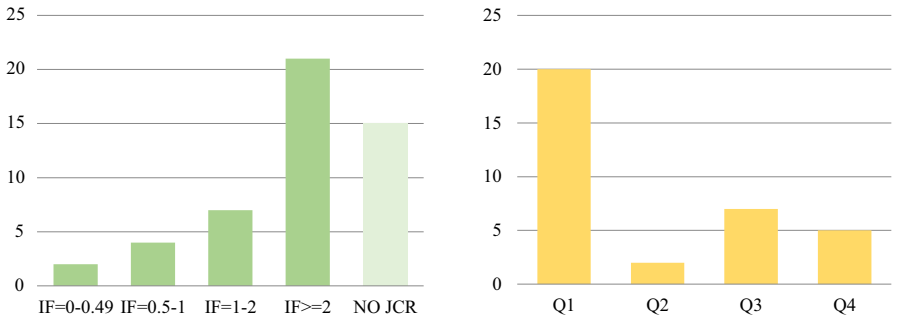

Fig. 6 Quality of the journals

has most activity in this field. As presented in Fig. 6, we evaluate the quality of the journal some features such as Q and IF. We investigate journal of published papers and journal of "system and software" is the most important journal according to some feature such as Q, IF and frequently published paper on design pattern detection. Moreover, the conferences are evaluated through some features such as Qualis, ERA (Fig. 6).

RQ1. What is the design pattern detection?

4 Design pattern detection

Software designers attempt to employ design patterns in software design phase, but design patterns may be not used in the implementation phase. Therefore, one of the challenging issues is conformance checking of source code and design, i.e., design patterns. Identifying design pattern from source code can help to achieve the design of an existing system as a reverse engineering task.

Design pattern detection (DPD) is an active field of research, useful to support software maintenance and reverse engineering. In particular, it gives useful hints to understand some design decisions (Chihada et al. 2015) and the detection of design patterns is a useful activity giving support to the comprehension and maintenance of software systems (Dwivedi et al. 2016). Several methods for design pattern detection from source codes have been proposed (Fig. 7).

RQ2: What kinds of design patterns are used in different publications?

5 Types of design patterns

In this section, different kinds of design patterns are reported that authors considered for evaluation and calculated precision and recall or F-measure. As aforementioned in Sect. 1 and Fig. 8, there are three categories of design patterns (Sebastian 2002) (These criteria will be explained in Sect. 10). Some papers have claimed that their method can detect large number of design patterns and reported the number of DPs which their method can detect but they may be false, positive or their methods may not be recognized by some of the patterns that have existed in sources. Table 4 represents most methods claiming to recognize

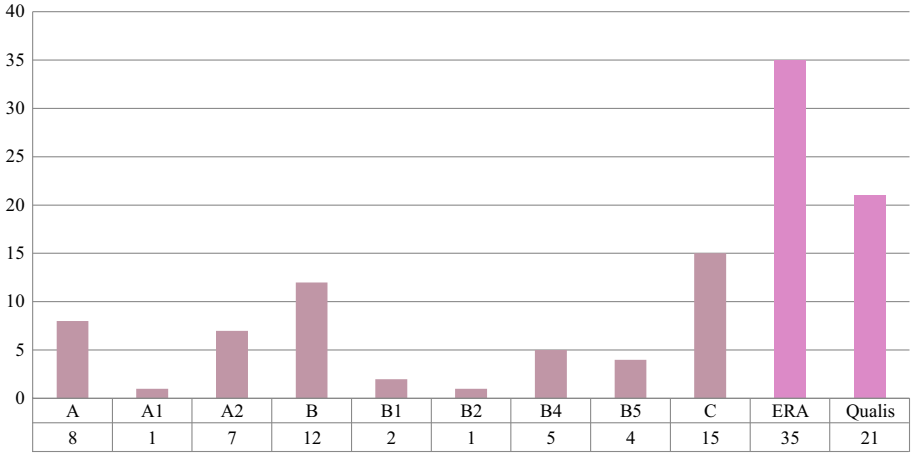

Fig. 7 Evaluation of conferences

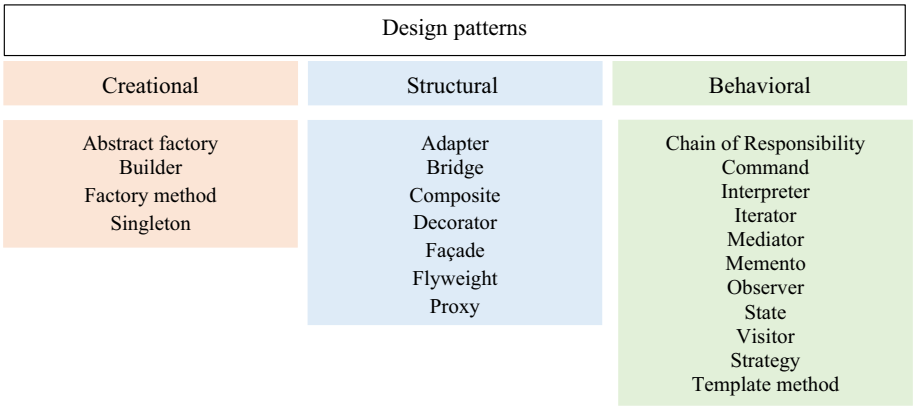

Fig. 8 Design patterns categories (GoF)

Table 4 Types of design patterns detected in papers

Design pattern type	References
Structural	Yu et al. (2018, 2013a), De Lucia et al. (2009a)
Behavioral	Thongrak and Vatanawood (2014), De Lucia et al. (2009b, 2015a, b), Dong et al. (2008a), Bernardi et al. (2015), De Lucia (2010a), Liu et al. 2018a, b)
Creational	Arcelli et al. (2008)
Structural and behavioral and creational (Group 1)	Fontana and Zanoni (2011), Chihada et al. (2015), Dwivedi et al. (2016, 2018), Mayvan and Rasoolzadegan (2017), Oruc et al. (2016), Di Martino and Esposito (2016), Yu et al. (2015), Alnusair et al. (2014), Bernardi et al. (2013, 2014), Lee et al. (2008), Elaasar et al. (2015), Romano et al. (2011), Rasool and Mäder (2011), Bernardi (2010), Guéhéneuc et al. (2010), Rasool and Mäder (2014), Alnusair and Zhao (2009, 2010), Hayashi et al. (2008), Uchiyama et al. (2011, 2014), Issaoui et al. (2015), Al-Obeidallah et al. (2017a, b), Qiu et al. (2010), Guéhéneuc and Antoniol (2008), Aladib and Lee (2018), Hussain et al. (2017, 2019), Thaller et al. (2019), Zhang et al. (2017)
Structural and behavioral (Group 2)	De Lucia et al. (2010b), Dong et al. (200b9), Alhusain et al. (2013a, b), Kniesel and Binun (2009a), Binun and Kniesel (2012a, b), Kniesel and Binun (2009b), Wang et al. (2012a)
Structural and creational (Group 3)	Kaczor et al. (2010), Dwivedi et al. (2016), Zanoni et al. (2015), Bashir et al. (2013)
Creational and behavioral (Group 4)	Ng et al. (2010)

a variety of design patterns. As shown in Fig. 9, in the highest level, most of the methods of different papers can detect all kinds of DPs. In the second level, the methods can detect structural and behavioral design patterns and in the lowest level, a few methods focus only on detection of structural DPs.

It should be noted that some design patterns have similar structures and some approaches cannot detect them such as Mediator, Abstract Factory and Façade or State,

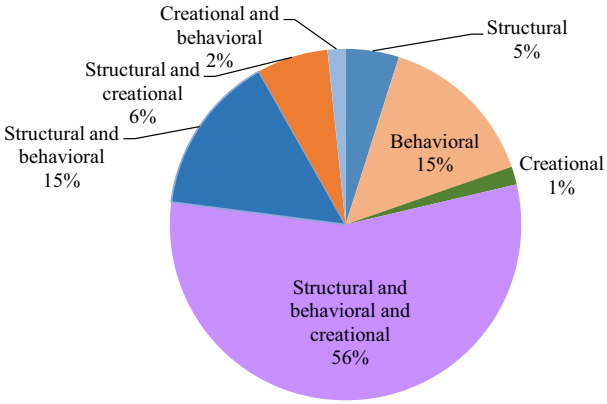

Fig. 9 The dispersion of the design patterns used

Strategy and Command or Builder, Bridge and Template Method.In Table 5, the differences and similarities of three design patterns are examined from different perspectives.

RQ3: What items are considered as inputs and which ones are most frequently used?

RQ4: What kinds of the input representations are used in the papers and which ones have been most used?

RQ5: What is the difference between some kinds of Data Representation such as ASG, AST, Feature vector and metrics?

RQ6: What basis the Different methods choose their input representation method?

6 Data representation

Each method receives a particular form of input and then converts the input into understandable format. Various items have been considered as inputs for example source code, graph and comments. Among the inputs, using source code had the most usage. Some papers have used items such as existing comments in runtime in order to detect the design patterns more accurate. It can be expected that in future, items related to input code log file, will be used to find out the design patterns. Some papers have not mentioned what input they have used and we refer to them as "Non-mention". There are various types of data representation such as feature vector, metric, class diagram, ontology knowledge, fact-based file, XML document, etc. Each method used a kind of data representation that provides appropriate data for detection process. Feature vector is a combination of several features that the presence or absence of these features is examined. If the feature exists, the number 1 is considered, otherwise, the zero number is considered. Metrics is used for quantitative methods that computes values for program metrics such as aggregation, generalization, association and so on. The difference between the metrics and the feature vector is that the vector considers as combination of presence or absence of features whereas in the metric for each feature spatial value is considered (feature vector is explained in Sect. 7.2). Other representations are Abstract

Table 5 The differences and similarities of three DPs

	Façade	Abstract factory	Mediator
Intent	Reducing interactions between objects	Creating a family of affiliated and related objects and hiding how to create them from the user	Simplify work with sub-systems
Delegation	All responsibilities assign to the facade object delegated to the main object	The responsibility for the construction is delegated to the factories	In this pattern design, there is an indirection
Coupling	The coupling is low and the changes of subsystems are hidden from the client's view	Classes are created using abstract factory. As a result, the coupling gets better	The connection between objects is reduced, resulting in low coupling
Cohesion	Communication among objects is deposited into the façade, which is the result of better cohesion	The responsibility for constructing objects is the responsibility of the abstract factory, which is the result of better cohesion	Using this design pattern, communications among objects are gathered in the main object, so cohesion gets better

Table 6 Type of input of design pattern detection methods

Type of input	References
Source code	Kaczor et al. (2010), Thankappan and Patil (2015), Fontana and Zanoni (2011), Wierda et al. (2008), Chihada et al. (2015), Dwivedi et al. (2016a, b, 2018), Yu et al. (2013a, b, 2015, 2018), De Lucia et al. (2009a, b, 2010, 2015a, b), Bernardi et al. (2015), De Lucia et al. (2010a), Liu et al. (2018a, b), Arcelli et al. (2008), Mayvan and Rasoolzadegan (2017), Oruc et al. (2016), Alnusair et al. (2014), Bernardi et al. (2013, 2014), Lee et al. (2008), Romano et al. (2011), Rasool and Mäder (2011, 2014), Bernardi (2010), Guéhéneuc et al. (2010), Alnusair and Zhao (2009, 2010), Hayashi et al. (2008), Uchiyama et al. (2011, 2014), Issaoui et al. (2015), Al-Obeidallah et al. (2017a, b, Qiu et al. (2010), Guéhéneuc and Antoniol (2008), Aladib and Lee (2018), Hussain et al. (2017, 2019), Thaller et al. (2019), Zhang et al. (2017), Dong et al. (2008b, 2009), Alhusain et al. (2013a, b, Binun and Kniesel (2012a, b), Wang et al. (2012a), Zanoni et al. (2015), Bashir et al. (2013), Ng et al. (2010), Dabain et al. (2015), Pradhan et al. (2015), Luitel et al. (2016), García et al. (2013), Kirasić and Basch (2008), Stencel and Wegrzynowicz (2008), Robinson and Bates (2016), Di Martino and Esposito (2013), Ren and Zhao (2012), Paydar and Kahani (2012), Gautam and Diwaker (2012), von Detten and Becker (2011), Rasool et al. (2010), Gupta et al. (2010a), Pande et al. (2010a, b), Chen and Qiu (2010), Han et al. (2009), Tripathi et al. (2009), Karam et al. (2014), Wang et al. (2012b), Zhu et al. (2009), Lebon and Tzerpos (2012), Fontana et al. (2011), Bouassida and Ben-Abdallah (2010), Von Detten and Platenius (2009), Wegrzynowicz and Stencel (2013), Stoianov and Şora (2010), Majtás (2011), von Detten (2011), Czibula and Czibula (2008), Chaturvedi et al. (2018), Alshira'h (2017)
Non-mention	Thongrak and Vatanawood (2014), Di Martino and Esposito (2016), Elaasar et al. (2015), Chaturvedi et al. (2016), Liamwiset and Wiwat (2013), Gupta et al. (2010b, 2011), Pande et al. (2010c), Panich and Vatanawood (2016), Gupta and Rao (2014), Gupta (2011, 2017), Bouassida and Ben-Abdallah (2009, 2010), Ba-Brahem and Qureshi (2014), Alnusair et al. (2013), Bayley and Zhu (2010)
Graph	Dong et al. (2008a)
Comments	Romano et al. (2011)

Syntax Tree (AST) and Abstract Semantic Graph (ASG). These parsing formats cause to high accuracy. AST is labeled and directed tree that can be used between a parser tree and data structure. ASG is a graph that has two nodes, element node and relation node. Element nodes represent elements such as attributes and classes. Relation nodes represent the relations such as aggregation and association. Some approaches convert source code and design pattern into rules and try to detect design pattern according to rules. Lattice is mathematical and abstract structure which is used for data representation for some approaches such as FCA based approach. As evident in Table 6 and Fig. 10, the most common types of input are source codes and as shown in the Table 7 and Fig. 11, the most common input representation is graph. Any approach uses a type of data representation that is consistent with its method. For instance, graph-based methods used graph representation to represent data, and ontology-based approaches use WebOntology Language (OWL). It is clear that the metric-based methods need metrics for design pattern detection, so, they represent input as metric format. It was witnessed in some of the papers that the input and the design pattern are represented in different formats.

This paper overviews different types of input. There are different types of input, but it is mostly considered to include source code. Most papers have one-phase for pre-representation, which is converted to the main representation when the input is converted into the intermediate representation. The phases will be examined in Sect. 7.

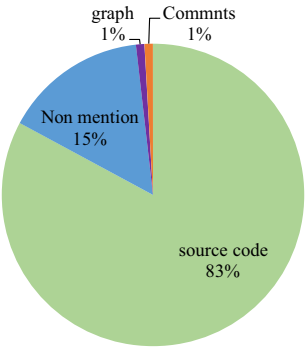

Fig. 10 Dispersing the types of input of design pattern detection methods

RQ7: What types of approaches are there in design pattern detection and which one of them have been most frequently used?

7 Overview of design pattern detection approaches

There are many research studies on design pattern detection, and the pattern detection approaches can be classified according to various indicators. In this paper, they have been classified according to their methods and solutions. Figure 12 represents a common way of design pattern detection approaches. A number of these approaches have a preprocessing phase, and in this phase the tasks like text mining, filtering and so on are done. This phase causes to reduce search space and high performance. As shown in Fig. 13, there are 13 approaches of design pattern detection an. Figure 14 represent dispersion of use approach, it represented the amount of using each approach in each year. Each number indicates the times of using an approach in each year, for example according to this figure, Graph-based is the most frequently used in 2010, 2011, 2013, and 2014. According to Fig. 15 learning-based approach among them is most frequently used; After that, graph-based, and reasoning-based approaches are, respectively, in the lower places in terms of use frequency. FCA-based, model checking approach, query-based and data fusion are less frequently used than the other approaches. In the rest of the paper, the variant of approaches is discussed.

7.1 Graph-based approach

Graph-based approach usually has the representation phase for the source code and the design patterns. As shown in Fig. 16, this approach uses various techniques such as graph isomorphism, graph matching, semantic matching, and cross correlation for design pattern detection. These methods usually have high accuracy and recall.

For example, in Pradhan et al. (2015), graph isomorphism has been used to find sub-graphs in the system graph using normalized cross-correlation, which provides a way to express the percentage of design patterns in the system. In another example, Bernardi et al. (2014) used graph matching and identified design patterns using matching between each design pattern and the overall system graph by annotating hierarchical elements with the informational role they play in each pattern. Yu et al. (2013b) detected the design patterns using graph isomorphism between system graph and design pattern graph. This method

Table 7 Type of data representation for design pattern detection method

Data representation	References/kinds of representation or input
Class diagram	De Lucia et al. (2009a, b, 2010, 2015a, b), Czibula and Czibula (2008) (source code), De Lucia (2010a) (source code), Rasool and Mäder (2014), Issaoui et al. (2015), Zhang et al. (2017), Bayley and Zhu (2010)
Sequence diagram	De Lucia et al. (2009b) (design pattern), Yu et al. (2018), Liu et al. (2018b), Karam et al. (2014), von Detten (2011), Bayley and Zhu (2010)
UML diagram	Di Martino and Esposito (2013) (software artifacts documentation), Ng et al. (2010)
Graph	Chihada et al. (2015), Yu et al. (2013a), Bernardi et al. (2013, 2014, 2015), Mayvan and Rasoolzadegan (2017), Oruc et al. (2016), Yu et al. (2015), Bernardi (2010), Pradhan et al. (2015), Yu et al. (2013b), Pande et al. (2010a, b, c, Karam et al. (2014), Majtás (2011), Chaturvedi et al. (2018), Liamwiset and Wiwat (2013), Gupta et al. (2010b, 2011), Gupta and Rao (2014), Gupta (2011, 2017), Ba-Brahem and Qureshi (2014)
Feature vector	Dwivedi et al. (2016a, b, 2018), Hussain et al. (2017, 2019), Thaller et al. (2019), Alhusain et al. (2013a)
Semantic structures	Zanoni et al. (2015), Robinson and Bates (2016)
Fact-based file	Dabain et al. (2015), Lebon and Tzerpos (2012) (fine-grained static facts), Ren and Zhao (2012) (source code fact data base)
Rules	Alnusair et al. (2014) (semantic web rule language), Thongrak and Vatanawood (2014) (semantic query-enhanced web rule language), Aladib and Lee (2018), Wang et al. (2012a), Rasool et al. (2010), Zhu et al. (2009)
AST (abstract syntax tree)	Kirasić and Basch (2008) (Input code), Fontana and Zanoni (2011), Arcelli et al. (2008), Lee et al. (2008), Bernardi (2010), Al-Obeidallah et al. (2017a, b, Stencel and Wegrzynowicz (2008), Paydar and Kahani (2012), von Detten and Becker (2011), Chen and Qiu (2010)
Ontology knowledge	Kirasić and Basch (2008) (Define patterns), Ren and Zhao (2012) (Rules in DP), Panich and Vatanawood (2016)) (resource description framework(RDF)/ontology web language (OWL) element), Alnusair and Zhao (2010) (ontology web language(owl)-description logic (DL) ontology), Alnusair and Zhao (2009), Di Martino and Esposito (2016) (object design ontology layer (ODOL) + OWL-S patterns representation), Di Martino and Esposito (2013) (object design ontology layer (ODOL) representation of software patterns), Alnusair et al. (2014) (ontology knowledge)
ASG (abstract semantic graph)	Lee et al. (2008), Qiu et al. (2010), von Detten (2011), Chaturvedi et al. (2016)
Matrix	Dong et al. (2008a, b, Gautam and Diwaker (2012) (Binary matrix), Romano et al. (2011) (a term-by-document), Dong et al. (2009b), Gupta et al. (2010a) (weighted matrix), Wang et al. (2012b)
Other representation	García et al. (2013) (linked list representation), Wegrzynowicz and Stencel (2013) (logic query), Rasool and Mäder (2011) (source code model) (a semi-formal XML-based definition of the patterns'structural features such as classes, their relationships, and method return types.), Thankappan and Patil (2015), Rasool et al. (2010)) (tree structure), Alhusain et al. (2013b) (feature subset), Guéhéneuc and Antoniol (2008) pattern and abstract level description language (PADL), Fontana et al. (2011) (Micro structure), Kaczor et al. (2010) (UML-like diagram), Luitel et al. (2016) (Answer set programming (ASP) facts), Liu et al. (2018a)
XML format	Rasool and Mäder (2014), Issaoui et al. (2015), Han et al. (2009) (XML feature type catalog), Di Martino and Esposito (2016) (XML metadata interchange (XMI) software artifact), Alshira'h (2017) (XML file), Zhang et al. (2017)

Table 7 (continued)

Data representation	References/kinds of representation or input
Metric	Guéhéneuc et al. (2010), Uchiyama et al. (2011, 2014), Dwivedi et al. (2018), Von Detten and Platenius (2009)
Concept lattice	Wierda et al. (2008), Tripathi et al. (2009)
Meta model	Elaasar et al. (2015) Visual pattern modeling language (VPML) meta model, Bashir et al. (2013), Wegrzynowicz and Stencel (2013)
Prolog	Hayashi et al. (2008), Binun and Kniesel (2012a, b), Stoianov and Şora (2010), Alnusair et al. (2013)
XML document	Bouassida and Ben-Abdallah (2009, 2010a, b)

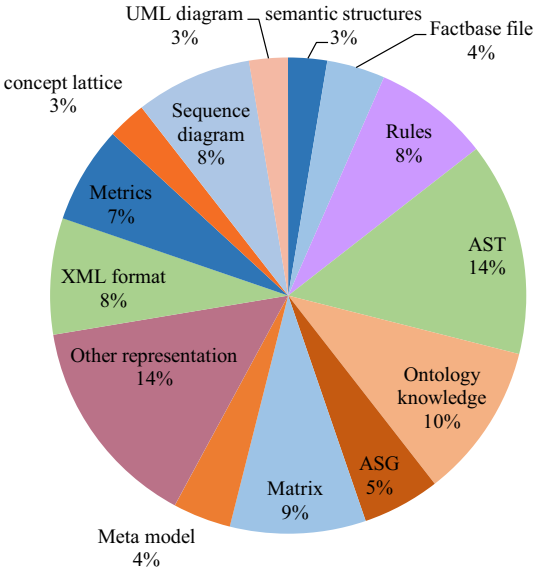

Fig. 11 Dispersing representation data methods used

identifies a number of design pattern samples and describes the relationships between classes in a diagram by weight assignment to graph edges.

7.2 Learning based approach

This approach can be divided into three categories: supervised, unsupervised and evolutionary, as shown in Fig. 17.

Machine learning has important role in design pattern detection field. We can make the models of design patterns and we can map design pattern detection problem into learning problem. Therefore, it causes to propose automatic approaches that they do not need an expert's help.

As mentioned, there are different implementations of design patterns and it makes it difficult to detect the design patterns from source codes, but if we try to model various implementations of design patterns, we can solve this challenge. According to the results

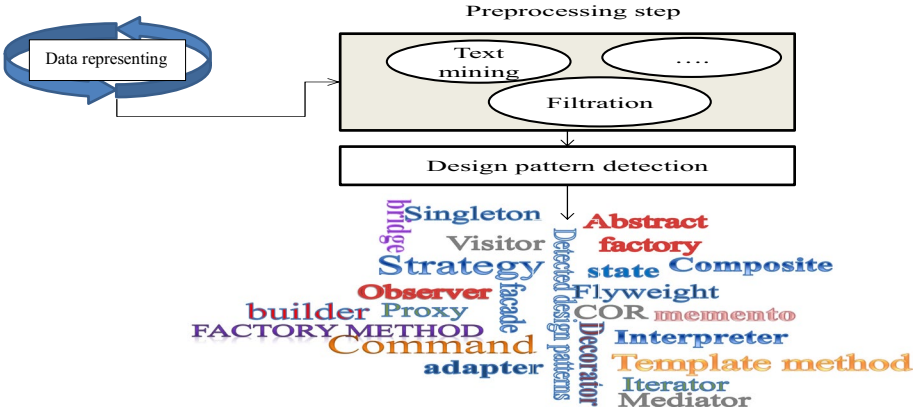

Fig. 12 Common way of design pattern detection methods

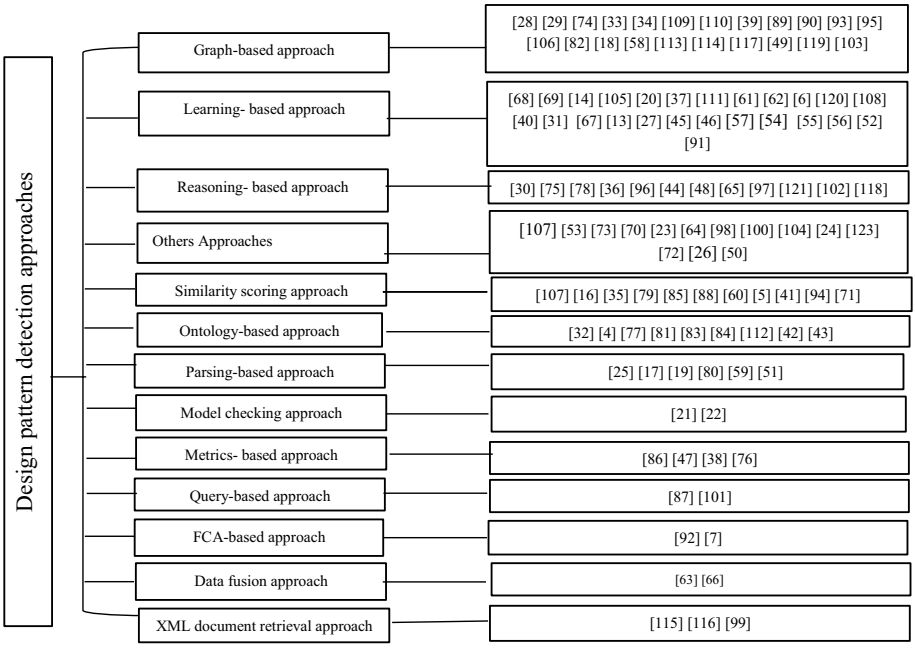

Fig. 13 Approaches of desing pattern detection methods

of the proposed methods that used machine learning to detect the design patterns can detect different implementations of DPs. In fact, machine learning and artificial intelligence can be used in different approaches and add to their intelligence and ability. For example, we can use machine learning to obtain some rules set as DP signatures. The signatures are so helpful for matching process. On the other hand, by tracking the signatures of design patterns, we can reduce the search space significantly. As a result, it also saves time and expenses. Machine learning helps to ensure that the proposed method is not dependent on input type and, over time, it gradually improves its learning with the use of training data,

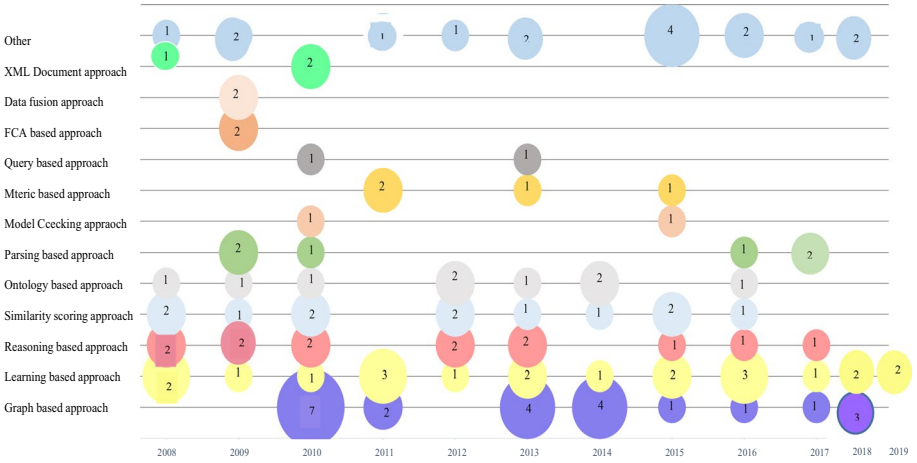

Fig. 14 Dispersion of used approaches per year

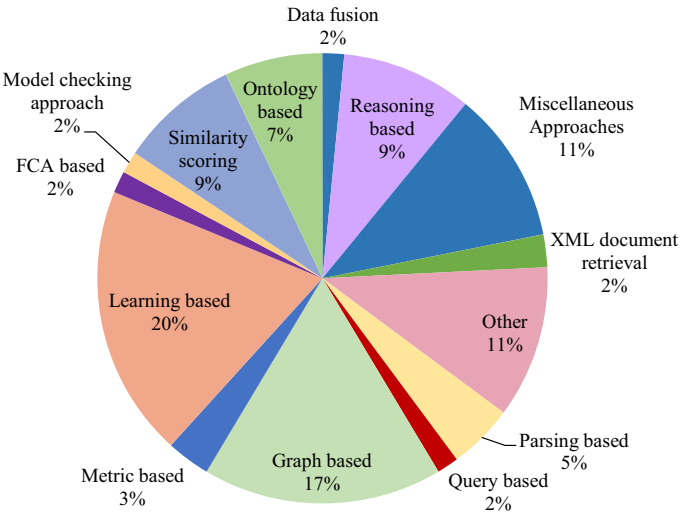

Fig. 15 Dispersion of used approaches

and its accuracy is improved by using artificial intelligence. Therefore, the proposed methods are automated.

As it is presented in Fig. 18 most of the presented methods have used the supervised learning to detect the design patterns. In supervised approach, some learning samples are needed among which 70% are considered for training and 30% for testing. Feature vector is a combination of several features. We can consider various properties as features. For example, properties of classes of DPs, such as structural and behavioral properties. In reality, we can consider structural and behavioral properties of DPs for detecting and it helps to discover them more accurately. For example, we can consider some properties such as

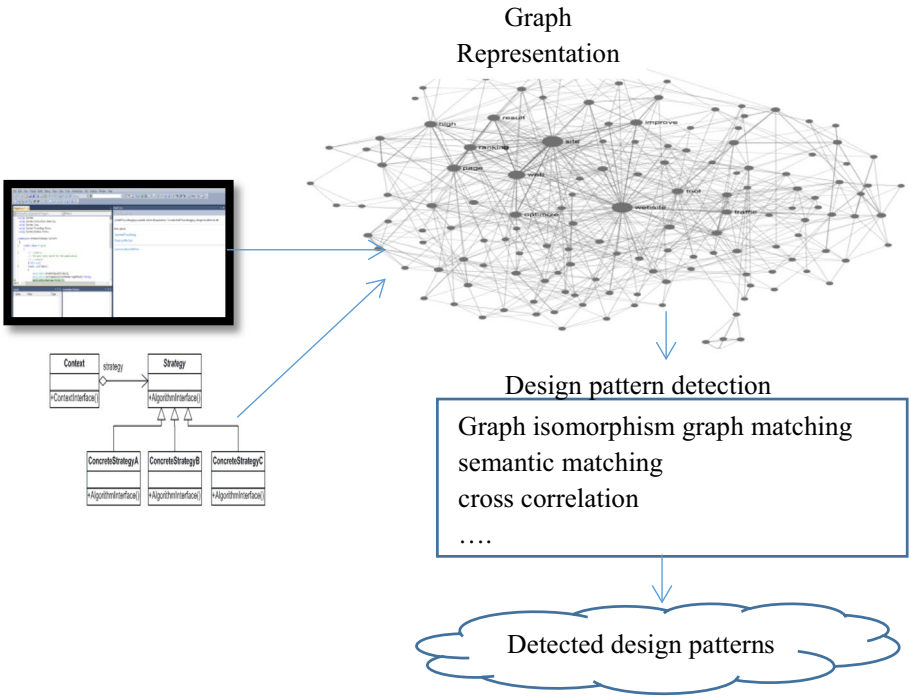

Fig. 16 Overview of graph-based approach

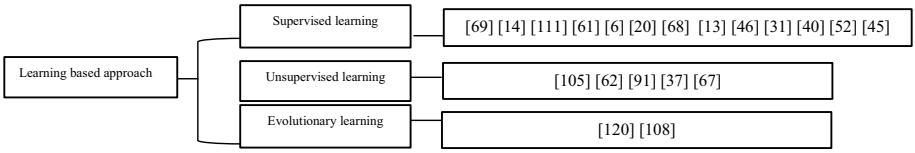

Fig. 17 Categories of learning based approach

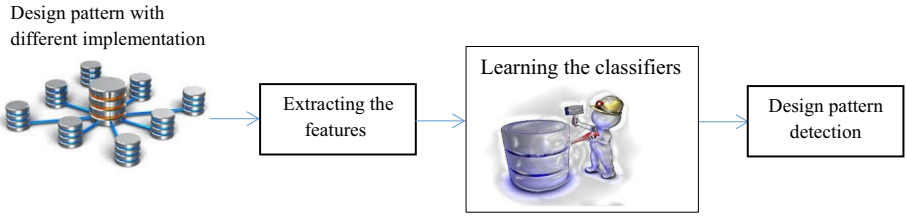

Fig. 18 Overview of learning based approach

the number of methods, number of children of classes as features or learning samples can include a combination of classes and relationships between program codes such as aggregation, delegation, inheritance and so on. For obtaining these features, at first, the source code and design pattern should be converted into class diagram or sequence diagram. In reality the detection of machine learning based approaches was done by matching, produced by machine learning.

In learning approach consider some feature such as aggregation, delegation, inheritance and so on. For obtaining these features, at first, the source code and design pattern should be converted into class diagram or sequence diagram.

Chihada et al. (2015) proposed a method using learning-based approach. This method converts class diagram of source code and design patterns to graphs. They have used object-oriented metrics as features and computed 45 features for each class of design patterns and created feature vectors. They trained various classifiers using the vectors then detected design pattern using them. The classifiers used in this method include K Nearest Neighbors (KNN), Support Vector Machine (SVM), C4.5, Simple logistic and SVM-PHGS. Another category of learning method is Evolutionary Learning (EA). Some examples of this category are Genetic algorithm, ant colony and so on. An evolutionary algorithm is a subset of evolutionary computation, a generic population-based meta-heuristic optimization algorithm. EA uses biological evolution such as reproduction, mutation, recombination, and selection. The evolutions of population obtain using fitness function and candidate solutions. Candidate solution used for optimization role of individual in a population and the quality of solution is determined by the fitness function. Gupta (2017) used GA and generated random mapping (population) between the system design and the design pattern graph. GA applied on random mapping for obtaining the best mapping based on their fitness function value.

7.3 Similarity scoring approach

Some papers used Similarity Scoring Approach (SSA) in which design patterns are detected using matching algorithms. Figure 19 represents an Overview of similarity scoring approach. As it is shown in this figure the system and design patterns are represented to intermediate representation such as graphs whose matching rate is measured by matching algorithms to detect the design patterns. In Dong et al. (2007), the matching algorithm was applied on the edges instead of vertices. This method, borrowed from machine learning, calculates the normalized cross-correlation between two matrices. In fact, this method calculates the cross-correlation between the part of code and the design pattern. They used semantic and behavioral analyses for reducing False Positives (FP).

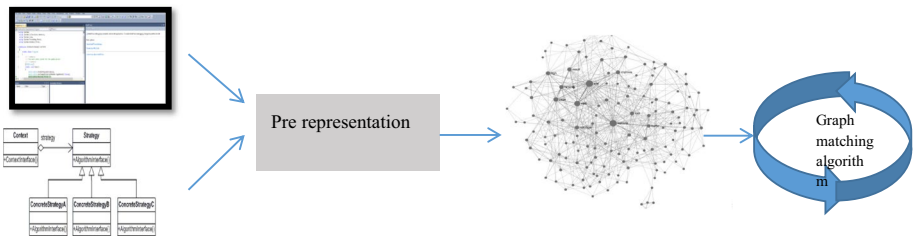

Fig. 19 Overview of similarity scoring approach

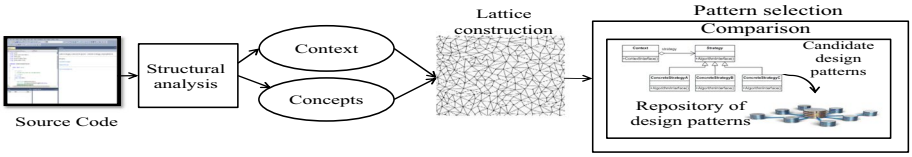

Fig. 20 Overview of FCA-based approach

7.4 FCA-based approach

The next approach, which is a branch of lattice theory, is formal concept analysis through which a set of elements which have common characteristics are identified. The methods used in this approach are based on logic and mathematics. In this approach, the collection of elements, attributes and binary relations between them is known as the context, and groups of elements which have common characteristics are called concepts. FCA-based approach is a mathematical method for identifying possible duplicate groups of formal objects with common formal features. Because most of these methods are able to identify design patterns with a certain number of roles, and while methods are meaningful. However, it does not necessarily provide the best results for us and has a high computational volume (Alnusair and Zhao 2010). As shown in Fig. 20, structural analysis is applied on the source code and then context and concepts are extracted. The next phase is lattice construction, which is an important representation for this approach. Afterwards, patterns selection phase is done by the comparison of candidate design patterns with the repository of design patterns.

For example, Tripathi et al. (2009) used FCA-based approach. This model is comprised of three phases including (1) the metadata generation, (2) software pattern detection by FCA, and (3) post process. The metadata generation is applied using the MOOSE tool (Chaturvedi et al. 2018). The MOOSE tool gives the input and generates language independent metadata. The elements, properties and context matrix are generated in the second phase from the metadata, and in this phase, concepts and lattice are generated by Concept Matrix based Concepts Generation (CMCG) algorithm. Finally, in the third phase, unconnected patterns are removed and the similar patterns are merged into the patterns generated by the second phase, and subsequently the neighborhood between the patterns is assigned.

7.5 Ontology-based approach

This approach uses structure source code knowledge. The ontology provides a common understanding of a concept. The ontology is the foundation of XML, Resource Description Framework (RDF) and Ontology Web Language (OWL). OWL is used as a tool for modeling knowledge of a particular domain. All existing knowledge in the domain is modeled as a family of affiliate concepts, in which there are interconnected relationships. In other words, all information is modeled as a hierarchical tree. Ontology-based methods use the semantic query to recognize design patterns.

As it is shown in Fig. 21, some methods which use ontology-based approach have pre-representation phase, after which the analyzer applies some analyses on the input and then ontologies are extracted. This approach uses reasoning and logic for design pattern detection.

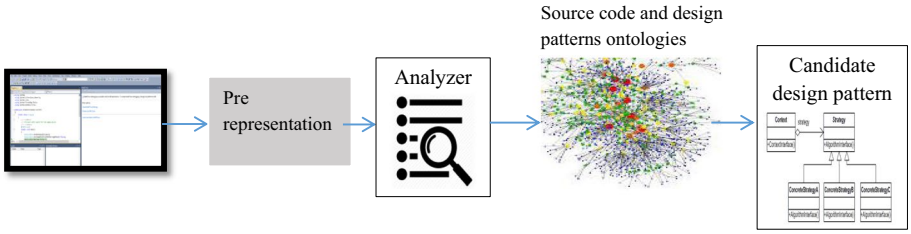

Fig. 21 Overview of ontology-based approach

Ontology-based methods are employed in Kirasić and Basch (2008). This method consists of three modules: (1) ontologies, (2) analyzer and (3) parser. The system gets source code as input and analyzer reports AST as the output. In Kirasić and Basch (2008), the source code, which is in the XML document format, is converted to AST by th parser.

7.6 Metrics-based approach

This approach uses different techniques for design pattern detection by making comparison between the metric values (such as aggregation, association, attributes and operations) and the expected values for design samples. Figure 22 represents phases of metric based approach. When source code and design patterns definition entered the metric generation phase and the metrics are generated, the metric values and the expected values for design samples are compared using different techniques.

For example, in Issaoui et al. (2015), filters were employed to search for semantic and structural symbols of design patterns samples. In this approach, design pattern filtering

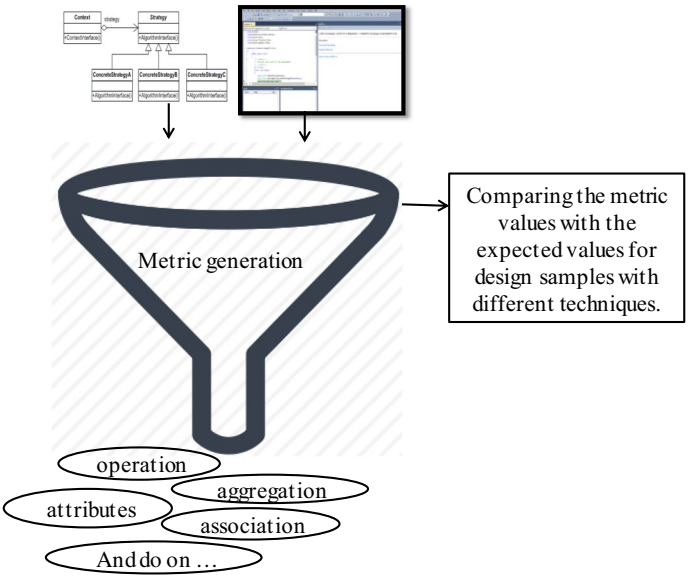

Fig. 22 Overview of metric-based approach

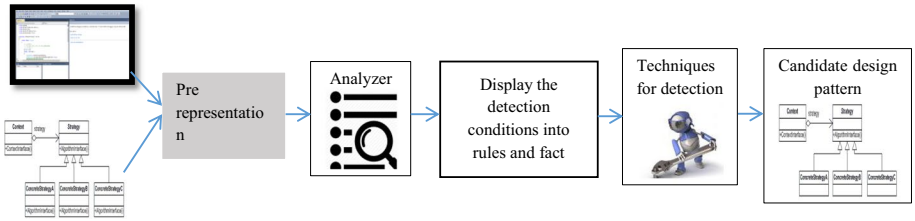

Fig. 23 Overview of reasoning-based approach

includes two phases: First, a semantic recognition uses the encoded meanings in the elements of design patterns. Moreover, in the second phase, a structural analysis is applied only when the first phase was successful in identifying of design patterns for the candidate. Each phase also uses metrics to determine a directory of instructions for design patterns in general, metric-based techniques are computationally efficient because of the reduced search space and their lower cost in comparison to the structural pattern recognition.

7.7 Reasoning-based approach

Reasoning based approach has two categories: logical and fuzzy reasoning. The researches that used reasoning approach, as shown in Fig. 23, firstly perform some analyses on the source code and the design patterns and afterwards, the detection conditions are displayed into rules that will be subsequently used for design pattern detection by various techniques (Mayvan and Rasoolzadegan 2017). In this approach, the Prolog and First Order Predicate Logic (FOPL) languages are used because they facilityate the condition adding and modification.

Al-Obeidallah et al. (2017a) proposed a method for design pattern detection that has high speed and precision and uses static and dynamic analysis. This method uses Prolog to display design patterns detection condition and uses backtracking mechanism and database function.

7.8 Parsing-based approach

Most of the research studies that used parsing-based approach usually employ parsing formats consisting of AST and ASG for data representation. This approach uses visual language parsing because it has high precision. According to Fig. 24, visual language grammar is applied on the design patterns, and finally, data representation phase is applied on the design patterns and the source code. It should be mentioned that this approach detects the design patterns by parsing phase.

In De Lucia et al. (2010b), visual language grammar was drawn upon for each design pattern and visual language parsing for design pattern detection. This method proposed an Eclipse plug-in for parsing and detection of the design pattern from the object-oriented source code (called Epad). This tool extracts the data model from source code and uses structural analysis to recover design pattern instances according to structural features. Moreover, this tool performs behavioral analyzes through the instrumentation and monitoring of the software system.

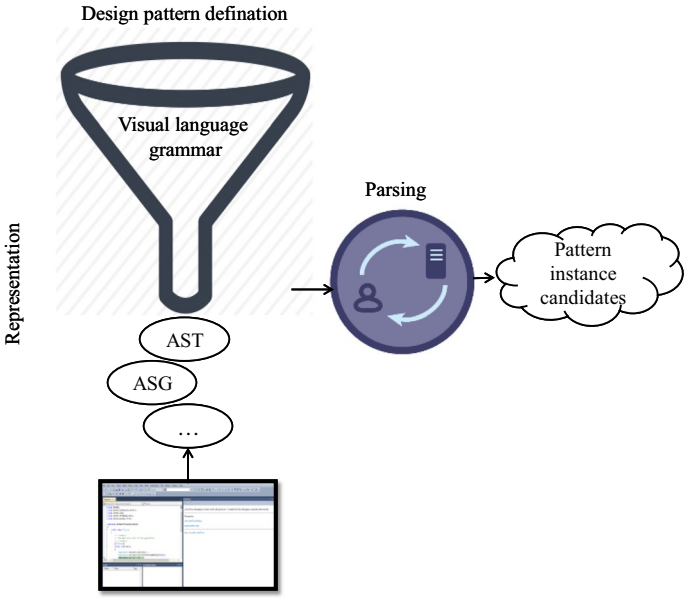

Fig. 24 Overview of parsing-based approach

7.9 XML document retrieval approach

A number of researches during 2008–2017 have used this approach. The approach adopts an XML document retrieval technique through which the UML diagram is transformed to the XML document. This process is feasible since the transformation of UML diagrams into XML documents is straightforward and can handle by all existing UML editors. As shown in Fig. 25, this approach considers the design pattern as an XML query, and the design as the target XML document where the pattern is searched. In XML document retrieval, the document can be considered as an ordered labeled tree.

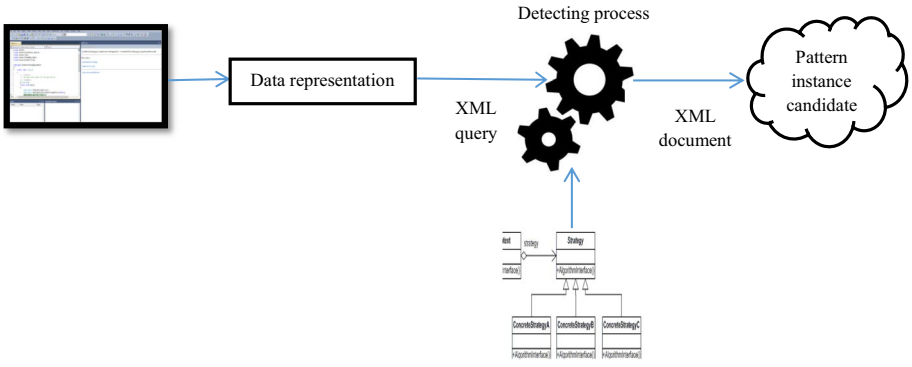

Fig. 25 Overview of XML document retrieval approach

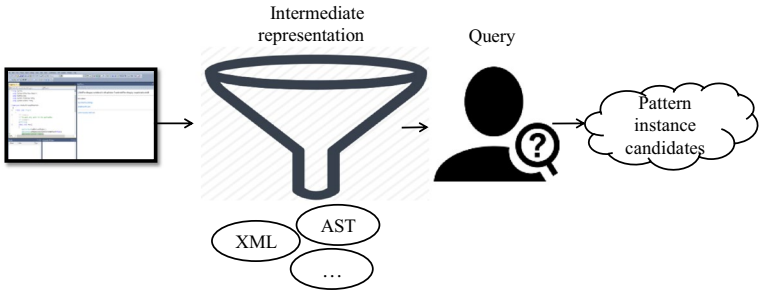

Fig. 26 Overview of query-based approach

Each node of the tree represents an XML element and the tree is analysed as a set of paths starting from the root to a leaf. Each query is examined as an extended query, that is, there can be an arbitrary number of intermediate nodes in the document for any parent–child node pair in the query. Documents that closely match the query structure with fewer additional nodes are given preference query is examined as an extended query, that is, there can be an arbitrary number of intermediate nodes in the document for any parent–child node pair in the query (Czibula and Czibula 2008).

Bouassida and Ben-Abdallah (2010b) used XML document retrieval approach for the pattern and the spoiled pattern detection– that is, it considers a design pattern, or spoiled pattern, as an XML query to be found in an XML document representing a design.

7.10 Query-based approach

This approach uses database queries to retrieve data from the relational database management systems. A number of researches used SQL queries, to extract different relationships from the source code model which was extracted in past. According to Fig. 26, intermediate representations such as AST, XML and so on are extracted from the source code and the design patterns are considered as queries. In this approach, design patterns are detected by the application of query to the information which exists in the intermediate representation.

Rasool et al. (2010) presented a recovery approach that uses regular expressions, SQL queries, and annotations. Regular expressions have a strong matching power, a simple syntax, and are flexible enough to extend the pattern specifications. Database queries are used for the retrieval of data from relational database management systems and annotations is employed to describe the semantics of the codes related to their pattern extraction approach.

7.11 Model checking approach

A number of papers used model checking approach. As it is shown in Fig. 27, structural analyses are applied on the source and byte code. Afterwards, some design pattern instances are candidate. In this approach, there is model checking approach that analyses the source code of the pattern instances to verify whether the behavioral definition of the pattern is satisfied.

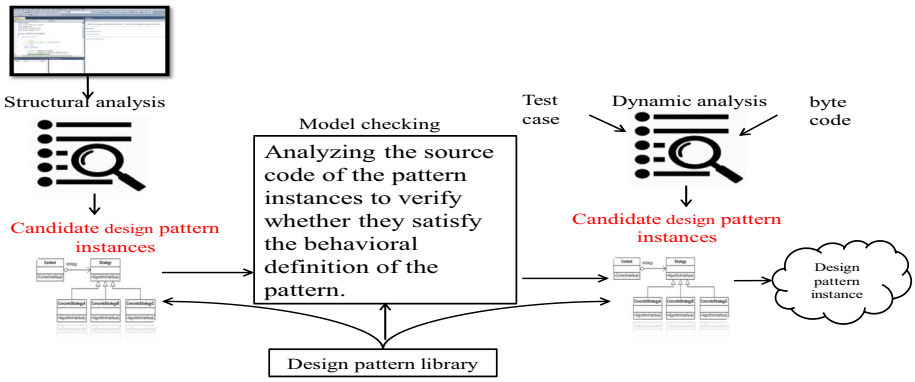

Fig. 27 Overview of model checking approach

De Lucia et al. (2010a) proposed a fully automatic approach that uses static and dynamic analyses for the detection design patterns. In this method, dynamic and static analyses, which analyse the interactions among objects using model checking were performed to verify the behavior design pattern instances. The dynamic analysis used code instrumentation and monitoring phase. In particular, this approach uses model checking to statically verify the behavioral expects of design pattern instances (Yu et al. 2015).

7.12 Data fusion approach

Another approach of DPD is data fusion approach. The reason for naming is they fusing the outputs of several DPD tools. Some papers used data fusion approach in the sense that they fused the outputs of several DPD tools to detect the design patterns. As shown in Fig. 28, this approach analyses the aspects of patterns by tools, and then, in the combining phase,

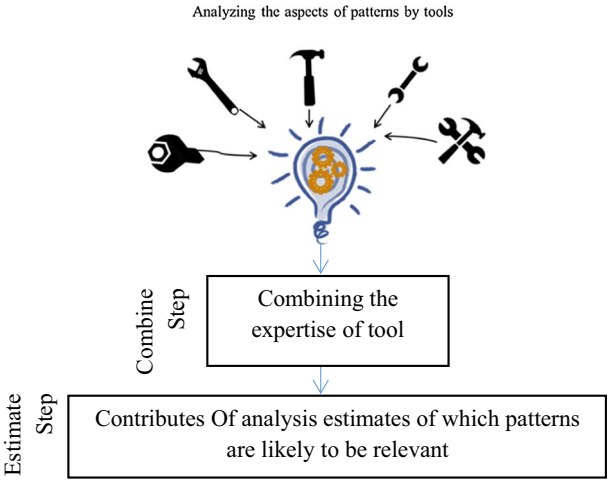

Fig. 28 Overview of query-based approach

the expertise of tools is combined. Finally, contributes of analysis estimates of which patterns are likely to be relevant (Kniesel and Binun 2009a).

Kniesel and Binun (2009a) reported what was achieved from a practical evaluation of DPD tools and describes how they underlie a novel approach to design pattern detection that is based on data fusion.

7.13 Others approaches

Besides, a number of researches have used approaches which are not included in any categories and are called as other approaches. Stephan and Cordy (2015) used clone detection approaches to detect design patterns using model clone detection. It presented how Model Clone Detection (MCD) approaches can be employed to detect patterns and antipatterns. Ideally, the notions discussed in this research will inspire further researches into using MCD or MCD-like approaches to improve the quality of Model-driven Engineering (MDE) software through direct detection of patterns and antipatterns in models. The proposed method in Stephan and Cordy (2015) defined and represented each pattern of interest as a model or a set of models and stored those patterns in their project and detected cross clones that pattern project and the projects of interest by running MCD tool.

Dabain et al. (2015) introduced a new approach called FINDER which consists of four stages. First, static relations are extracted from Java class files using Javex and QL. Second, Java source files are parsed using a tool called Variable Explorer developed to extract additional relations which are not available in the first stage, such as method invocation on collections. In the third stage, QL is used to integrate the facts from the first two stages and unify them in one fact-based file. Finally, the fact-based file is filtered for candidates which satisfy the static definition of design patterns to be detected. The tool performs static analysis using well-defined scripts which correspond to the fine-grained design pattern detection rules.

Fontana et al. (2011) identified four microstructures which may be seen as the building blocks of the construction of design motifs, and the indicators for the presence of design motifs during the detection process. The focus of this paper was on the description of the microarchitecture which was found from re-engineering of the various implementations of design motifs. They focused on the architectures of design patterns and their roles. Some of the microarchitectures, for example, EDPs, clues, sub-patterns are significant for the identification of design pattern architecture, and there are also some micro architectures which are relevant to role identification.

In Guéhéneuc et al. (2010), the authors used a combination of the constraint programming and the metrics-based approach. They supplemented their previous work with numerical analysis and improved its performance.

Guéhéneuc and Antoniol (2008) represented multilayered semiautomatic approach for design pattern detection from Java source code. This was an approach to semi-automatically identify microarchitectures which are similar to the design motifs in the source code and to ensure the traceability of these microarchitectures between implementation and the design. This approach detected relationships using static analysis and dynamic analysis based on trace analysis to compute exclusivity and lifetime relationships for aggregation and composition relationships. DEMIMA (Guéhéneuc and Antoniol 2008) consists of three layers. Two layers recover an abstract model of the source code including binary class relationships, and the other layer identifies design patterns in the abstract model.

Design pattern detection approaches are divided into 13 categories. Some approaches are used less and some more. Among all, graph-based approach is most popular, and over the years, many cases have been added to it, which in turn, increases the performance of this approach.

RQ8:What are the advantages of each approach?

RQ9: What are the disadvantages of each approach?

8 Advantages and disadvantages of approaches

Table 8 represents the advantages and disadvantages of different approaches. The approaches are evaluated with regards to many factors including large test capability, *P*(precision), *R*(recall), the ability to discover singleton and the design patterns that have similar structures such as strategy and state, reducing search space, evaluation on large test cases and pre-representation. Some approach didn't report values for some factors and we assigned them unknown values and we represented these cases by "–".

According to this report, the graph-based approach shows good result, but since it has not been tested with large cases, there is no assurance of its results when tested with large cases. The learning based approach, with its high *P, R*, can detect all design pattern implementations, but is not able to distinguish between state and strategy in majority of cases. The parsing based approaches have a high precision, even though they were negatively evaluated for the large cases DPs. While the metric-based approach has positive results for our items, its precision and recall are low. The efficiency of some approaches such as ontology based, query based, XML document retrieval, and model checking approach is high, but is dependent on the selection of the test cases. The precision and recall of these approaches are shown as "Not clear". Some approaches did not report any amount for precision and recall; hence, the result of their ability to recognize Singleton, State, and strategy cannot be reported and we represented these cases by "–". Among these approaches, FCA based approach is less commonly used to recognize patterns of design. The multi-layered approach, as an instance of others approaches, was also evaluated because this approach produced good results and therefore, was frequently cited over the recent years. Regarding the reported cases, parsing based, learning based and graph-based approaches have high precision and recall. Our suggestion in using these approaches is using combination of graph based and learning based approach, because machine learning is developing and so widespread that can help to detecting DPs and graph-based approach can helpful for learning based approach to detect DPs that learning based cannot detect them solely.

RQ10: What are the datasets which are used in different papers for evaluation and which ones have been most frequently used?

Table 8 Advantages and disadvantages of different approaches of approaches

Nos.	Approach	Evaluation on large test cases	Precision, recall	Singleton detection	State and strategy detection	Reducing search space	Pre-representation
1	Graph-based approach	×	High recall Low precision	√ [except Yu et al. (2013a), Chaturvedi et al. (2018)]	√ [except Yu et al. (2013a), Oruc et al. (2016), Chaturvedi et al. (2018)]	√	√
2	Learning-based approach	√	High precision High recall	√[except Fontana and Zanoni (2011), Chihada et al. (2015), Dwivedi et al. (2016, 2018)]	×[except Dwivedi et al. (2016), Uchiyama et al. (2014, 2011), Dwivedi et al. (2018)]	√	√
3	Parsing-based approach	×	High precision	×[except Aladib and Lee (2018)]	×[except De Lucia et al. (2010b)]	×	√
4	Query-based approach	√	Not clear	–	–	√	√
5	Ontology-based approach	×	Not clear	×[except Alnusair et al. (2014)]	√[except Thongrak and Vatanawood (2014)]	×	√
6	Reasoning-based approach	×	Low precision Low recall	√	√	×	√
7	Similarity scoring approach	×	Not clear.	×	×	×	√
8	FCA based approach	×	–	–	–	×	√
9	XML document retrieval approach	–	Not clear	–	–	×	√
10	Data fusion approach	√	High precision High recall (for decorator, observer and visitor)	×	×	–	–
11	Model checking approach	×	Not clear	√	√	×	√
12	Metric based approach	×	Low precision Low recall	√	√	√	√
13	Others approaches (multilayered approach)	√	Low precision	√	√	×	–

Table 9 Types of datasets that used by DPD methods

Dataset	References
JUnit (www.junit.org)	Mayvan and Rasoolzadegan (2017) (v3.8,v4.1), Zanoni et al. (2015) (v3.7), Yu et al. (2015) (v3.8), Pradhan et al. (2015),17, Alnusair et al. (2014) (v3.7), Bernardi et al. (2014) (v 3.7), Bernardi et al. (2013) (V 3.7), Chihada et al. (2015), Dong et al. (2008a, b, Paydar and Kahani (2012) (v 3.7), Romano et al. (2011) (v 3.7), Rasool and Mäder (2011) (v 3.7), von Detten (2011) (v 4.8.2), Bernardi (2010) (v 3.7), Rasool and Mäder (2014), Dong et al. (2009b) (v 3.7), Wang et al. (2012b) (v 4.7), Alnusair and Zhao (2009) (v 3.7), Uchiyama et al. (2011) (v 4.5), Uchiyama et al. (2014) (v 4.5), Kaczor et al. (2010) (v3.7), Issaoui et al. (2015) (v 3.7), Issaoui et al. (2016) (v 3.7), Kniesel and Binun (2009a) (v 3.7), Al-Obeidallah et al. (2017a), Nagy and Kovari (2015), Qiu et al. (2010), Fontana et al. (2011) (v 3.7), Kniesel and Binun (2009b) (v 3.7), Guéhéneuc et al. (2010) (v 3.7), Guéhéneuc and Antoniol (2008) (v 3.7), Yu et al. (2018), Liu et al. (2018a, b, Oruc et al. (2016), Dwivedi et al. (2018), Al-Obeidallah et al. (2017b), Thaller et al. (2019), Zhang et al. (2017)
JHotDraw (www.jhotdraw.org)	Mayvan and Rasoolzadegan (2017), Dwivedi et al. (2016) (v 5.1,v 7.0.6), Di Martino and Esposito (2016) (v 5.1), Zanoni et al. (2015) (v 5.1), Yu et al. (2015) (v 5.1), Alnusair et al. (2014) (v5.1), Bernardi et al. (2014) (v 5.1), Bernardi et al. (2014) (v 7), Bernardi et al. (2013) (v 5.1), De Lucia et al. (2009b), Stencel and Wegrzynowicz (2008), Dong et al. (2008b) (v 5.1), Chihada et al. (2015), Dong et al. (2008a), García et al. (2013) (v 5.2), Yu et al. (2013a), Paydar and Kahani (2012) (v 5.1), Romano et al. (2011) (v 5.1), Rasool and Mäder (2011) (v 5.1), Rasool et al. (2010) (v 6.1.2), De Lucia et al. (2010a) (v 5.1), De Lucia et al. (2010b) (v 5.1), Bernardi (2010), Dong et al. (2009b), Han et al. (2009) (v 6.0), De Lucia et al. (2009a) (v 5.1, v 6.0b1), Rasool and Mäder (2014) (v 5.1), Wang et al. (2012b) (v 6.0), Alnusair and Zhao (2010) (V. 6.0B1), Alnusair and Zhao (2009) (v 6.0), Alhusain et al. (2013a) (v 5.1), Alhusain et al. (2013b) (v 5.1), Kaczor et al. (2010) (v 5.1), Bouassida and Ben-Abdallah (2009), Issaoui et al. (2016) (v 5.1), De Lucia et al. (2015a) (v 5.1), Ng et al. (2010) (v 5.4b1), Kniesel and Binun (2009a) (v 5.1,v 0.6), Al-Obeidallah et al. (2017a), Binun and Kniesel (2012a) (v 5.1,v 6.0), Binun and Kniesel (2012b) (v 5.1, v 6.0), Tramontana (2014), (v 0.6), Qiu et al. (2010), Fontana et al. (2011) (v 5.1), Kniesel and Binun (2009b) (v 5.1, v 0.6), Wegrzynowicz and Stencel (2013) (v 60b1), De Lucia et al. (2015b) (v 5.1), Stoianov and Şora (2010) (v 6.0b1), Guéhéneuc et al. (2010) (v 5.1), Guéhéneuc and Antoniol (2008) (v 5.1), Yu et al. (2018), Liu et al. (2018a), b), Aladib and Lee (2018), Dwivedi et al. (2018), Al-Obeidallah et al. (2017b), Thaller et al. (2019), Zhang et al. (2017), Chaturvedi et al. (2018)
JRefactory (www.jrefactory.sourceforge.net)	Mayvan and Rasoolzadegan (2017), Zanoni et al. (2015) (v 2.6.24), Alnusair et al. (2014) (v 2.6.24), Bernardi et al. (2014) (v 2.6.24), Chihada et al. (2015), García et al. (2013), Dong et al. (2008b) (v 2.6.24), Yu et al. (2013a), Paydar and Kahani (2012) (v 2.6.64), Rasool and Mäder (2011) (v 2.6.64), von Detten (2011) (v 2.6.64), De Lucia et al. (2010a) (v 2.6.64), Rasool and Mäder (2014) (v 2.6.24), Wang et al. (2012b) (v 2.9.18), Issaoui et al. (2016) (v 1.0), Ng et al. (2010) (v 2.6.24), Kniesel and Binun (2009a) (v 2.6, v 2.8), Al-Obeidallah et al. (2017a), Qiu et al. (2010), Fontana et al. (2011) (v 2.6.24), Kniesel and Binun (2009b) (v 2.6,2.8), Guéhéneuc and Antoniol (2008) (v 2.6.34), Yu et al. (2018), Al-Obeidallah et al. (2017b), Thaller et al. (2019), Zhang et al. (2017)

Table 9 (continued)

Dataset	References
Java AWT (www.java.sun.com/j2se/1.5.0/docs/guide/awt/index.html)	Oruc et al. (2016) (v 1.3), Yu et al. (2015) (v 5.0), Dong et al. (2008a), Alnusair and Zhao (2009), Dong et al. (2009b) (v 1.6), Kniesel and Binun (2009a), Binun and Kniesel (2012a) (v 1.3), Binun and Kniesel (2012b) (v 1.3), Stoianov and Şora (2010) (v 1.3)
QuickUML (www.quj.sourceforge.net)	Zanoni et al. (2015), (2001), Bernardi et al. (2014) (v 2.1), Bernardi et al. (2013) (v 2.1), Bernardi et al. (2015) (v 2.1), Romano et al. (2011) (2001), De Lucia et al. (2009a), Rasool and Mäder (2014) (v 2,001),Kaczor et al. (2010) (2001), Ng et al. (2010) (2001), Kniesel and Binun (2009a) (2001), Al-Obeidallah et al. (2017a), Fontana et al. (2011) (2001), Kniesel and Binun (2009b) (2001), De Lucia et al. (2015b) (2001), Guéhéneuc and Antoniol (2008) (2001), Dwivedi et al. (2018), Al-Obeidallah et al. (2017b), Thaller et al. (2019), Zhang et al. (2017)
Lexi (www.iro.umontreal.ca/~labgelo/p-mart/index.php)	Zanoni et al. (2015) (v0.1.1 alpha), Pradhan et al. (2015) (alpha), Bernardi et al. (2014) (v0.1), Bernardi et al. (2013) (V 0.1), Kniesel and Binun (2009a) (v 0.1 alpha), Fontana et al. (2011) (v 0.1.1), Kniesel and Binun (2009b) (v 0.1), Guéhéneuc et al. (2010) (v 0.1.1 alpha), Liu et al. (2018a, b), Thaller et al. (2019), Zhang et al. (2017)
Apache Ant (www.ant.apache.org)	(Rasool et al. (2010) (v 1.6.2), De Lucia et al. (2009a), Rasool and Mäder (2014) (v 1.6.2), Stoianov and Şora (2010) (v 1.6.2)
MapperXML (www.iro.umontreal.ca/~labgelo/p-mart/index.php)	Romano et al. (2011) (v 1.9.7), De Lucia et al. (2015b) (v 1.9.7), Guéhéneuc and Antoniol (2008) (v 1.9.7), Fontana et al. (2011) (v 1.9.7), Thaller et al. (2019), Zhang et al. (2017), Zanoni et al. (2015)
Nutch (www.iro.umontreal.ca/~labgelo/p-mart/index.php)	Bernardi et al. (2014) (v 0.4), Bernardi et al. (2013) (v 0.4), Fontana et al. (2011) (v 0.4), Zanoni et al. (2015) (v 0.4), Thaller et al. (2019), Zhang et al. (2017)
PMD (www.iro.umontreal.ca/~labgelo/p-mart/index.php)	Bernardi et al. (2014) (V1.8), Bernardi et al. (2013) (v 1.8), Ng et al. (2010) (v 8.1), Kniesel and Binun (2009a) (v 1.8), Fontana et al. (2011) (v 1.8), Kniesel and Binun (2009b) (v 1.8) Zanoni et al. (2015) (v 0.8), Thaller et al. (2019), Zhang et al. (2017)
Java.io	Stoianov and Şora (2010) (v 1.4.2), Wang et al. (2012a) (v 1.5), Kniesel and Binun (2009a) (v1.4), Binun and Kniesel (2012a, b)
Group 1	
Eclipse IDE in the version 2.1	Von Detten and Platenius (2009)
Active object concurrency pattern (Lavender and Schmidt (1995)), Master Workers Pattern or the Divide and Conquer Pattern (Mattson et al. (2004))	Di Martino and Esposito (2013)
org.jhotdraw_gures. (a package that responds for graphical figures of the JHotdraw framework. (v6.0 beta 1))	Wang et al. (2012a)
Java.net	Stoianov and Şora (2010) (v 1.4.2)
Sharp Develop (CASE tool), Paint.Net	
(software for picture modi_cation)	Majtás (2011)
4 test cases	Thongrak and Vatanawood (2014)
Getting Things Done (GTD)-Free (http://gtd-free.sourceforge.net/)	Dabain et al. (2015)
DPExample	Zanoni et al. (2015)

Table 9 (continued)

Dataset	References
Eiffel systems (www.gobosoft.com/eiffel/gobo/index.html)	Lebon and Tzerpos (2012)
GALib++(www.lancet.mit.edu/ga/)	Rasool and Mäder (2014) (v 2.4)
Libg++(ftp.gnu.org/gnu/libg++/)	Rasool and Mäder (2014) (v 2.7.2)
Common Component Modeling Example	
(CoCoME) (http://www.cocome.org/downloads.html)	von Detten and Becker (2011)
Informa	Pradhan et al. (2015)
lizzy(v1.1.1),Hodoku(v2.1.1) (http://hodoku.sourceforge.net/en/index.php),barcod4g(v2.1.0),RstpProxy(v3.0) (https://sourceforge.net/p/iptv2rtspproxy/wiki/Home/),	
Teamcenter(v6.02.199) (https://sourceforge.net/projects/tceav/)	Yu et al. (2015)
Jrat (jrat.sourceforge.net)	Pradhan et al. (2015)
Voldemort(open source java software system) (http://www.projectvoldemort.com/voldemort/developer.html)	Bernardi et al. (2014) (v 1.3.x)
ApacheAvro(open source java software system) (https://avro.apache.org/)	Bernardi et al. (2014) (V 1.6)
JDT(open source java software system) (https://projects.eclipse.org/projects/eclipse.jdt/developer)	Bernardi et al. (2014) (V3.6.1)
33 industrial components	Guéhéneuc and Antoniol (2008)
PURE Toolkit, JINI-based Home Appliance	
System, Project Management Supporting	
Tool, MP3 Player	Lee et al. (2008)
Applied Java Patterns(AJP)	Stencel and Wegrzynowicz (2008)
CDK (cdk.sourceforge.net)	Dong et al. (2009b)
GFP (gfd.sourceforge.net)	Dong et al. (2009b)
Eclipse JDT (http://www.eclipse.org/jdt/)	De Lucia et al. (2009a)
Grizzly and RIP Worker, subsystems of the	
controller	Wierda et al. (2008)
mockup a case study	
from the UML diagrams from Freeman et al. (2004)	Panich and Vatanawood (2016)
Squiggle (Open Source LAN Messenger)	

Table 9 (continued)

Dataset	References
(http://squiggle.codeplex.com)	Bashir et al. (2013)
Two java programs	Hayashi et al. (2008)
Dresden OCL(Object Constraint Language)	Ng et al. (2010) (v 1.1)
TROVE (v 1.1b5) (https://sourceforge.net/projects/trove4j/), GANTT PROJECT (v 1.10.2) (https://sourceforge.net/projects/ganttproject/), AZUREUS (v 2.3.0.6) (https://sourceforge.net/projects/azureus/)	Kaczor et al. (2010)
All 23 design patterns in the GoF catalog	Bayley and Zhu (2010)
GANTT PROJECT(v1.10.2)	
(ganttproject.sourceforge.net), HOLUBSQL	
(v1.0) (www.holub.com/software/holubSQL/),JSETTLERS(v1.0.5)(jsettlers.sourceforge.net), JTANS (v1.0)(jtans.sourceforge.net), RISK	
(v1.0.7.5) (javarisk.sourceforge.net)	Guéhéneuc et al. (2010)
Subgroup 1	Elaasar et al. (2015)
Group 2	
JEdit (www.jedit.org)	Yu et al. (2013a), Dong et al. (2009b)
DOM4j (https://dom4j.github.io/)	Yu et al. (2013a, 2015) (v 1.6.1), Yu et al. (2018)
Eclipse platform	Kniesel and Binun (2009b) (v 3.1), Kniesel and Binun (2009a)
Swing (https://sourceforge.net/projects/swinguieditor/)	De Lucia et al. (2009a), Stoianov and Şora (2010) (v 1.4)
the set of programs where patterns are,	
applied in small-scale programs (60 in total),	
large-scale programs (158 in total from the	
Java library 1.6.0_13, Spring Framework 2.5	
RC2)	Uchiyama et al. (2011, 2014)
JUZZLE (juzzle.sourceforge.net)	Kaczor et al. (2010) (V 0.5), Guéhéneuc et al. (2010) (v 0.5)
Netbeans (www.iro.umontreal.ca/~labgelo/p-mart/index.php)	
	Zanoni et al. (2015) (v1.0.x), Fontana et al. (2011) (v 1.0), Thaller et al. (2019)
Log4g (http://logging.apache.org/log4j/)	Bernardi et al. (2014) (v 1.2.15), Dong et al. (2008b)
Apache Tomcat (v 4.1.37) (http://tomcat.apache.org/)	Kniesel and Binun (2009a, b)

Table 9 (continued)

Dataset	References
ArgoUML (v 0.18.0) (https://sourceforge.net/projects/argouml/), Team Core	Binun and Kniesel (2012a, b)
Subgroup 2	Fontana and Zanoni (2011), Arcelli et al. (2008)

9 Types of datasets

In this section, the datasets used in papers on design pattern detection are shown (Table 9 and Fig. 29). Junit, JHotDraw and JRefactory were most frequently used. In the second place are datasets likeQuickUML and JavaAWT. As mentioned in Guéhéneuc (2007), PMART is a pattern-like micro architecture repository, which includes some datasets like JHotDrow, JUnit, JRefactory, QuickUML, Lexi, Netbeans, MapperXML, Nutch and PMD.

Some papers had no evaluation. A number of papers used small case studies and it is not enough to confirm that they can detect design patterns properly. It is unclear whether their methods can detect design pattern in large systems or not. For example, Thongrak and Vatanawood (2014) just was tested with small cases. A number of papers were tested on real systems such as Czibula and Czibula (2008). Another disadvantage found with the papers was that they were not tested on open source systems, Yu et al. (2013b) as an instance. Some papers considered the dataset and report no precision, recall or F-measure, or they reported another subject. For example, Yu et al. (2013b) examined their method for recognizing the metrics on special dataset and their goal was to obtain something, which is necessary for design pattern detection. Given these limitations, the most popular design patterns are reported in the next section for evaluation and comparison.

In the Fig. 29, the datasets used only once are called Group 1, and thedatasets used only twice are referred to as Group2. Group1 is comprised of DPExample, Lizzy,

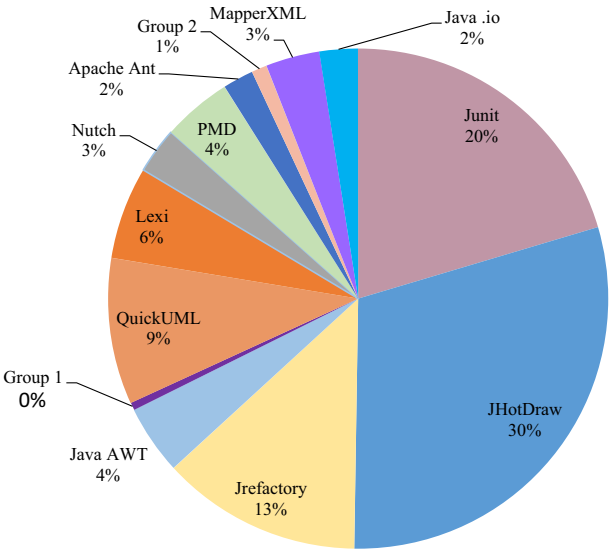

Fig. 29 Distribution of datasets used in DPD methods

Hodoku, barcod4g, RstpProxy, Teamcenter, Jrat, Informa, Voldemort, JDT, Eclipse JDT, 33 industrial systems, PURE Toolkit, JINI-basedHome Appliance System, Project Management Supporting Tool, MP3 Player, Applied Java Patterns(AJP), CoCoME, CDK, GFP, Grizzly, RIP Worker subsystems of the controller, mock up a case study from the UML diagrams, GALib++, Libg++, Squiggle, Two java programs, Dresden OCL, TROVE, GANTT PROJECT, AZUREUS,, Eiffel systems, Eclipse IDE, ApacheAvro, Active Object Concurrency Pattern, Master Workers Pattern or the Divide and Conquer Pattern, org. jhotdraw_gures, Java.net, sharp develop (case tool) Paint.Net, Four test cases, GoFcatalog, GANTT PROJECT, HOLUBSQL, JSETTLERS, JTANS, RISK, 2 java program, GTE-Free and sub group2. Subgroup 1 includes 2 case study; the first case study involves the specification of 11 GoF design patterns for UML (a general-purpose modeling language) and detecting their occurrences in a large UML design model of an open-source software project and the second case study involves the specification of ten CF design patterns for BPMN (a DSML) and detecting their occurrences in a large BPMN model designed for the financial services industry.

Group 2 includes 30 open source projects, Apache Tomcat, Netbeans, Log4g, JEdit, DOM4j, 2test case, ArgoUML, TeamCore, Eclipse platform, JUZZLE, Swing and sungroup 2. Subgroup 2 includes AF-earthlink, AF-java.net, net, Jboot, AF-WikiPedia, AF-rice, AF-vico.org, AF-itec, JVending-Registry-CDC, AF-c-sharpcorner, AF-u_ycat, AF-apwebco, AF-javapractises, AF-Gamma, Dynamicdispatcher, ehcache, Fdsapi, Sunxacml, Doubletype, bexee, wasa, wfmopen, GroboUtils, Nodal, Rambutan, Sparql, Infovis, fipaos, mandarax, JasperReports, Batik.

In response to RQ7, itcan be said that JHotDrow, JRefactory, JUnit, Java AWT, QuickUML, Lexi, Netbeans, Nutch, PMD, MapperXML, Apache Ant. Group1, and Group2 are used for evaluation in different papers. JHotDrow was the one most used for evaluation across different papers, while some datasets were used only once or twice. Some features of JHotDrow version 5.1 are provided in Table 10.

RQ11: What are the conclusion and comparison of the quantitative results of different approaches?

RQ12: which approaches have high precision and recall? What are our suggestions in using one of these techniques?

10 Quantitative results

In this section, the quantitative results of studies which used the most popular datasets including JhotDrow, Jrefactory, Junit, and PMD are reported. The quantitative results of papers from 2008 to 2018 have been considered. In the Tables 11, 12, 13, 14, measures of

Table 10 Some features of JHotDrow version 5.1

Number of components	Number of classes	Number of methods	Number of lines of codes
11	155	1316	8876

Table 11 Results of JhotDrow

Design pattern	Alhusain et al. (2013a)	Chaturvedi et al. (2016)	Mayvan and Rasoolzadegan (2017)	Dwivedi et al. (2016)	Di Martino and Esposito (2016)	Yu et al. (2015)	De Lucia et al. (2009b)	Guéhéneuc and Antoniol (2008)	Binun and Kniesel (2012a)
	P, R, F-M	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*
JhotDrow									
Singleton	–	1, 1, 1	1, 1, 1	–	–	1, –, –	–	1, –, –	–
Factory method	–	–, 0, –	1, 1, 1	–	1, 1, 0.8	1, –, –	–	0.01, –, –	–
Abstract factory	–	–	–	LRNN 1 ,1, 1 Decision tree 1, 0.91, 0.95	–	–	–	–	–
Composite	0.47, 1, 0.58	1, 1, 1	1, 1, 1	–	0.5, 1, 0.66	1, –, –	–	0.33, –, –	1, 0.67, 0.8
Adapter	0.11, 0.15, 0.13	0.85, 1, 0.91	1, –, –	LRNN 1 ,1, 1 Decision tree 1 ,1, 1	1, 0.9, 0.94	–	–	0.03, –, –	–
Decorator	1, 1, 1	1, 0.33, 0.49	1, 1, 1	–	1, 1, 1	1, –, –	–	0.07, –, –	1, 0.89, 0.94
Template method	–	0.8, 1, 0.88	1, 1, 1	–	1, 1, 1	1, 1, 1	–	0.06, –, –	–
Observer	0.90, 0.66, 0.76	–, 0, –	1, –, –	–	1, 1, 1	1, –, –	0.55, 1, 0.7	0.25, –, –	1, 0.65, 0.78
Visitor	–	1, 0.5, 0.66	–	–	–	1, –, –	–	1, –, –	–
Strategy	–	0.86, 1, 0.92	1, –, –	–	0.92, 0.92 0.92	–	–	0.28, –, –	–
State	–	0.86, 1, 0.92	1, –, –	–	0.8, 0.89, 0.84	–	–	0.8, –, –	–
Command	0.16, 0.69, 026	0.89, 0.89,, 0.89	–	–	1, 1, 1	–	–	0.09, –, –	–
Bridge	–	0.8, 1, 0.88	–	–	–	–	–	–	–
Façade	–	1, 1, 1	–	–	–	–	–	–	–
Prototype	–	1, 1, 1	–	–	0.5, 1, 0.66	–	–	1, –, –	–
Flyweight	–	1, 1, 1	–	–	–	–	–	–	–
Proxy	1, 1, 1	–	–	–	–	–	–	–	1, 0.96, 0.97
COR	–	–	–	–	–	–	–	–	1, 0.92, 0.95

Table 11 (continued)

Design pattern	Alnusair and Zhao (2009)	Bernardi (2010)	De Lucia et al. (2010a)	Al-Obeidallah et al. (2016)	Bernardi et al. (2013)	Dong et al. (2009b)	Kniesel and Binun (2009a)	De Lucia et al. (2015b)	Kaczor et al. (2010)
	P, R, F-M	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*
Composite		1, 1, 1	–	1, –, –	0.91, 0.97, 0.94	1, –, –	–	–	0.5, 1, 0..67
Adapter	0, –, –	0.14, 1, 0.24	–	1, –, –	0.93, 0.84, 0.88	1, –, –	0.76, 0.60, 0.51	–	–
Template method		–	1, 1, 1	–	0.76, 0.92, 0.93	–	–	–	–
Observer		1, 1, 1	0.55, 1, 0.7	–	0.91, 1, 0.95	–	0.76, 0.60, 0.67	0.86, 0.75, 0.80	–
Visitor		1, 1, 1	1, 1, 1	–	–	–	0.73 0.80, 0.76	–	–
Strategy		–	0.46, 1, 0.63	–	0.83, 1, 0.9	0.9, –, –	–	0.71, 0.71, 0.71	–
State		0.33, 0.88, 0.48	0.05, 1, 0.1	–	0.72, 1, 0.83	–	–, –, –	0.62, 1, 0.76	–
Bridge		–	–	0.93, –, –	–	–	0, 0, 0	–	–
Proxy		–	–	1, –, –	0.94, 0.89, 0.91	–	–	–	–
Singleton	0, –, –	0.67, 0.67, 0.67	–	–	1, 1, 1	–	–	–	–
Prototype		0.13, 0.5, 0.20	–	–	0.88, 0.85, 0.86	–	–	–	–
Abstract factory		0.4, 1, 0.57	–	–	–	–	–	–	0, 0, 0
Command		1, 1, 1	1, 1, 1	–	0.88, 0.88, 0.88	–	–	–	–
Decorator		–	–	1, –, –	0.87, 0.87, 0.87	–	0.77, 1, 0.87	–	–

Design pattern	Binun and Kniesel (2012b) (v5)	Binun and Kniesel (2012b) (v6.1)	Liu et al. (2018a)	Dwivedi et al. (2018)	Liu et al. (2018a)
	P, R, F-M	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*
Composite	1, 1, 1	1, 1, 1	–	1, 1, 1	1, 1, 1
Adapter	–	–	–	1, 1, 1	0.42, 1, 0.59
Template method	–	–	–	1, 1, 1	–
Observer	1, 0.5, 0.66	1, 1, 1	1, 1, 1	–	–

Table 11 (continued)

Design pattern	Binun and Kniesel (2012b) (v5)	Binun and Kniesel (2012b) (v6.1)	Liu et al. (2018a)	Dwivedi et al. (2018)	Liu et al. (2018a)
	P, R, F-M	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*
Strategy	–	–	1, 1, 1	–	–
State	–	–	1, 1, 1	–	–
Abstract factory	–	–	–	1, 0.78, 0.88	–
Proxy	1, 1, 1	1, 1, 1	–	–	–
COR	–	1, 1, 1	–	–	–
Decorator	1, 1, 1	1, 0.95, 0.97	–	–	1, 0.33, 0.49
Bridge	–	–	–	0.99, 1, 0.99	0.71, 1, 0.83
Singleton	–	–	–	–	1, 1, 1
Prototype	–	–	–	–	1, 1, 1
Facade	–	–	–	–	1, 1, 1
Flyweght	–	–	–	–	1, 1, 1
Command	–	–	–	–	0.67, 0.89, 0.76

precision (*P*), recall (*R*) and *F-measure* for the proposed method in each paper is reported. In these papers, three parameters (viz., precision, recall and F-measure) are used for evaluating. The equations for these parameters are Eqs. (1), (2) and (3). In fact, *TP* is the number of design patterns which were correctly identified by various methods in papers. *FP* is the number of design pattern instances that do not belong to a design pattern, but the methods incorrectly identified them as belonging to that design pattern. *FN* is the number of design pattern instances which belong to a design pattern, but the methods have not identified them as belonging to that design pattern. *FP* is the number of design pattern instances which do not belong to a design pattern but are incorrectly identified them. Equation (3) shows the equal for *F-measure* that is a combination of *P* and *R*. Some papers did not report *F-Measure* therefore, the criteria were calculated using Eq (3).

In Figs. 30, 31, 32 and 33, the average *P*, *R* and *F-Measure* are reported for each design pattern. In PMD all of the 23 design patterns including abstract factory, state, prototype, bridge, facade, flyweight, Chain Of Responsibility (COR), interpreter, mediator and memento are not considered in papers. The papers which used the other datasets consider 23 design patterns (GOF). Therefore, it can be concluded that, according to our reports, there is less effort to detect design patterns in PMD. In most papers, state and strategy are affiliated, therefore, the methods that could not identify the state were not able to recognize the strategy, and vice versa. In most papers, the adapter was well identified as shown in the Fig. 32. Most papers focused on design patterns such as singleton, factory method, composite, adapter, decorator, template method, observer, visitor, strategy, state, command and builder, and as shown in 30 and 31, the COR, flyweight and the proxy had the same results, and their values were simultaneously increasing and decreasing, such as state and strategy. Further, it can be said that singleton has been detected with a high *F-Measure* and in most of papers, composite, adapter and decorator not only were tested more than the other patterns, but also provided better results. According to these figures, design patterns such as flyweight, proxy, Chain Of Responsibility(COR), interpreter, mediator and memento have the highest P, R and F-measure, i.e., close to 100%. However, caution must be exercised because these design patterns were used in a small number of papers, and the other design patterns which produced lower quantitative results were evaluated in a larger number of papers.

$$P = \frac{\sum_{i=1}^{|c|} TP_i}{\sum_{i=1}^{|c|} TP_i + FP_i} \tag{1}$$

$$R = \frac{\sum_{i=1}^{|c|} TP_i}{\sum_{i=1}^{|c|} TP_i + FN_i} \tag{2}$$

$$F\text{-}Measure = \frac{2 \times P \times R}{(P + R)} \tag{3}$$

As shown in Fig. 30, some design patterns including builder, memento, mediator, iterator and interpreter were not considered for evaluation. COR, flyweight and proxy have the highest F-measure among the others, bridge have the lowest F-measure.

As indicated in Fig. 31, factory method, bridge, visitor, command, prototype, façade, flyweight, proxy, COR, interpreter, mediator, memento and builder have the highest F-measure (100%).

As shown in Fig. 32, composite has the highest *P* and *R and F-measure* (100%).

Figure 33 suggests that singleton has the highest *P, R* and *F-measure*, and some design patterns such as singleton, itrator, proxy, strategy, template method, decorator and adaper have the highest result for *R* and command has lowest result for recall.

Some design pattern such as Observer and visitor were considered in large numbers of papers, but the other design patterns such as command, singleton and strategy considered few papers.In most papers, State and strategy are affiliated with each other, and the methods which were not able to identify the state were unable to recognize the strategy, as well, and vice versa.In most papers, the singleton was well identified. Most papers focused on certain design patterns like singleton, factory method, composite, adapter, decorator, template method, observer, visitor, strategy, command and builder. Among them, the composite, observer and decorator, proxy had the same results, and their values were simultaneously increasing and decreasing, as is the case with state and strategy.In other words, singleton was detected with a high percentage, while composite, adapter, and decorator were tested more than others and produced better results. The datasets that considered design patterns like façade, flyweight, proxy, COR, interpreter, iterator, mediator and memento have the highest *P, R* and *F-measure*, that is close to 100%. It is noteworthy that these design patterns were considered in a small number of papers, and those with lower results were more frequently evaluated.

RQ13: What are the open issues for the future and what can be done to improve performance of design pattern detection?

11 Open issues

Although there have been many advances in this field and researchers have worked on the design pattern detection for many years, there are still problems to be dealt with and as a result, the field cannot be considered fully develop yet.

Among weaknesses which there are in most papers, one of them is the fact that most of the papers tested the methods on small test cases that have few number of classes and lines of code thus, there is no guarantee they can work well with large cases and it is not enough to confirm that they can detect design patterns properly. It is unclear whether their methods can detect design pattern in large systems or not. Consequently, methods that can detect design patterns in large and real cases are needed. For example, Dwivedi et al. (2016) was just tested with one code. Oruc et al. (2016) developed an automated detection tool called DesPaD. DesPaD's performance might suffer from large-sized projects. This method tested on cases that represented in Table 15.

This method can be improved by the analysis of the software metrics of a given source code. This problem of methods with large cases can be improved by filtering of the source code, as mentioned in Yu et al. (2015). In the future, the methods are expected to filter the template before integrating specific cases to speed up the implementation of the approach when dealing with large-scale software systems also the clustering and techniques such as text mining and using appropriate semantics of annotations and graph decomposition can reduce the search space and making the application of the classifier feasible task (Chihada et al. 2015; Zanoni et al. 2015; Rasool et al. 2010). It is expect

Table 12 Results of JUnit

Design pattern	Uchiyama et al. (2011)	Alnusair and Zhao (2009)	Mayvan and Rasoolzadegan (2017)	Oruc et al. (2016)		Dwivedi et al. (2016)		Yu et al. (2015)	Uchiyama et al. (2014)	Dwivedi et al. (2018)	Chihada et al. (2015)
				V 3.8	V 4.1						
	P, R, F-M	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*		*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*
JUnit											
Singleton	0.63, 1, 0.77	0, 0, 0	–	–	0.25, 1, 0.40	LRNN	1, 1, 1	1, 1, 1	–	–	–
						Random forest	1, 1, 1				
Factory method	–	–	1, 1, 1	–	–	–		1, 1, 1	–	–	–
Abstract factory	–	–	–	–	–	LRNN	0.83.3, 1, 0.9	–	–	1, 1, 1	–
						Random forest	1, 1, 1				
Composite	–	1, –, –	1, 1, 1	1, 1, 1	1, 1, 1	–		1, 1, 1	–	–	–
Adapter	1, 0. 6, 0.75	–	1, 1, 1	1, 1, 1	–	LRNN	1, 1, 1	1, 1, 1	1, 0.75, 0.85	1, 1, 1	–
						Random forest	1, 1, 1				
Decorator	–	–	1, 1, 1	1, 1, 1	1, 0.25, 0.4	–		1, 1, 1	–	–	–
Template method	0.71, 0.83, 0.76	1, –, –	1, 1, 1	1, 1, 1	0.91, 1, 0.95	LRNN	1, 1, 1	1, 1, 1	0.86, 1, 0.92	1, 1, 1	–
						Random forest	1, 1, 1				
Observer	–	1, –, –	1, 1, 1	–	–	–		1, 0.5, 0.67	–	–	1, 1, 1
Strategy	0.31, 0.8, 0.92	–	1, 1, 1	–	–	–		–	0.5, 1, 0.66	–	–
State	0.4, 0.66, 0.49	–	1, 1, 1	–	–	–		–	0.67, 1, 80.8	1, 1, 0.99	–
Bridge	–	–	–	1, 1, 1	0.25, 1, 0.4	1, 0.9, 0.94		–	–	–	–
Itrator	–	–	–	–	–	–		–	–	–	1, 1, 1

Table 12 (continued)

Design pattern	Al-Obeidallah et al. (2017b)	Bernardi et al. (2014)	Issaoui et al. (2015)	Guéhéneuc and Antoniol (2008)	Bernardi (2010)	Alnusair et al. (2014)	Chaturvedi et al. (2016)	Al-Obeidallah et al. (2017a)	Bernardi et al. (2013)
	P, R, F-M	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*
Singleton	–	1, 1, 1	1, 1, 1	1, –, –	1, 1, 1	–	1, 1, 1	1, 1, 1	0.97, 1, 0.98
Factory method	1, 1, 1	–	1, 1, 1	0, –,–	–	–	1, 1, 1	1, 1, 1	–
Abstract factory	–	–	1, 1, 1	0, –, –	1, 1, 1	–	1, 1, 1	1, 1, 1	–
Composite	–, 0, –	1, 1, 1	1, 1, 1	1, –, –	1, 1, 1	1, 1, 1	1, 0, 0	1, 0, 0	0.97, 0.85, 0.9
Adapter	0.71, 0. 71, 0. 71	1, 1, 1	1, 1, 1	0, –,–	1, 1, 1	1, 0.33, 0. 49	1, 0.45, 0. 62	1, 0.45, 0.62	–
Decorator	1, 1, 1	0.8, 1, 0.88	1, 1, 1	1, –, –	–	1, 1, 1	1, 1, 1	1, 1, 1	0.97, 0.81, 0.88
Template method	–	0.79, 0. 86, 0.82	1, 1, 1	0, –,–	–	–	0.5, 1, 0.66	0.5, 1, 0.66	–
Observer	–	1, 1, 1	1, 1, 1	0.25, –, –	0.5, 0.67, 0.57	–	1, 0, 0	1, 0, 0	0.92, 0.79, 0.85
Strategy	–	1, 1, 1	1, 1, 1	0, –,–		–	1, 1, 1	1, 1, 1	–
State	–	–	1, 1, 1	0, –,–	0.67, 0.67, 0.67	–	1, 1, 1	1, 1, 1	–
memento	–	1, 1, 1	–	–	–	–	1, 1, 1	–	–
Itrator	–, 0, –	1, 1, 1	–	–	–	–	1, 0, 0	1, 0, 0	0.83, 0.83, 0.83
Command	–	0.92, 0. 95, 0. 93	1, 1, 1	0, –,–	1, 1, 1	–	1, 1, 1	1, 1, 1	–
Visitor	–	–	1, 1, 1	0.5, –,–	1, 1, 1	–	1, 1, 1	1, 1, 1	–
Prototype	–	–	1, 1, 1	–	0.5, 1, 0.66	–	1, 1, 1	1, 1, 1	–
Builder	–	–	–	–	–	–	1, 1, 1	1, 1, 1	–

Table 12 (continued)

Design pattern	Al-Obeidallah et al. (2017b)	Bernardi et al. (2014)	Issaoui et al. (2015)	Guéhéneuc and Antoniol (2008)	Bernardi (2010)	Alnusair et al. (2014)	Chaturvedi et al. (2016)	Al-Obeidallah et al. (2017a)	Bernardi et al. (2013)
	P, R, F-M	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*
Bridge	–	–	–	–	–	–	1, 1, 1	1, 1, 1	–
Facade	–	–	–	–	–	–	1, 1, 1	1, 1, 1	–
Flyweight	–	–	–	–	–	–	1, 1, 1	1, 1, 1	–
Proxy	–	–	–	–	–	–	1, 1, 1	1, 1, 1	–
COR	–	–	–	–	–	–	1, 1, 1	1, 1, 1	–
Interpreter	–	–	–	–	–	–	1, 1, 1	1, 1, 1	–
Mediator	–	–	–	–	–	–	1, 1, 1	1, 1, 1	–

Table 13 Results of Jrefactory

Design pattern	Chaturvedi et al. (2016)	Mayvan and Rasoolzadegan (2017)	Guéhéneuc and Antoniol (2008)	Chihada et al. (2015)	Alnusair et al. (2014)	Al-Obeidallah et al. (2017a)	De Lucia et al. (2010a)	Bernardi et al. (2014)	Al-Obeidallah et al. (2017b)
	P, R, F-M	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*	*P, R, F-M*
JRefactory									
Singleton	1, 0.83, 0.9	1, –, –	0.14, –, –	–	0.57, –, –	1, 0.83, 0.90	–	1, 1, 1	1, 1, 1
Factory method	0.98, 0.91, 0.94	1, –, –	0.01, –,–	–	0.6, –,–	0.98, 0.91, 0.94	–	1, 1, 1	1, 0.67, 0.80
Abstract factory	–	–	0, –, –	–	0, –, –	–	–	1, 1, 1	–, 0, –
Composite	–	–	1, –, –	–	–	–	–	–	1, 1, 1
Adapter	1, 0.94, 0.96	1, 1, 1	0.36, –, –	0.76, 0.76, 0.76	00.42, –, –	1, 0.94, 0.97	–	0.97, 0.78, 0.86	1, 0.83, 0.80
Decorator	1, 1, 1	1, –, –	–, 0, –	–	–, 0, –	1, 1, 1	–	–	1, 0.67, 0.8
Template method	0.19, 1, 0.31	1, –, –	–, 0, –	–	–	0.19, 1, 0.32	1, –, –	0.85, 1, 0.91	–
Observer	–, –, –	1, –, –	–, 0, –	–	–	–, –, –	1, –, –	1, 1, 1	–
Visitor	1, 0.5, 0.66	1, –, –	0.5, –,–	–	–	1, 0.5, 0.66	0.5, –, –	0.99, 0.96, 0.97	–
Strategy	1, 0.73, 0.84	1, –, –	0.09, –, –	–	–	1, 0.73, 0.84	1, –, –	1, 0.98, 0.98	–
State	1, 0.73, 0.84	1, –, –	0.09, –, –	–	–	1, 0.73, 0.84	1, –, –	0.88, 0.88, 0.88	–
Command	0.69, 0.88, 0.77	–	–, 0, –	–	–	0.69, 0.88, 0.77	1, –, –	–	1, 0.94, 0.96
Prototype	–	–	1, –, –	–	–	–	–	–	
Builder	–, 0, –	–	–	–	–	–, 0, –	–	1, 1, 1	–, 0, –
Façade	1, –, –	–	–	–	–	1, –, –	–	–	1, 1, 1
Proxy	–	–	–	–	–	–	–	0.94, 1, 0.97	1, 0.5, 0.66
Memento	–	–	–	–	–	–	–	1, 0.91, 0.95	1, 1, 1
Flyweight	–	–	–	–	–	–	–	–	1, 1, 1
Bridge	–	–	–	–	–	–	–	–	–, 0, –

Table 14 Results of PMD

Design pattern	Bernardi et al. (2014) *P, R, F-M*	Bernardi et al. (2013) *P, R, F-M*	Kniesel and Binun (2009a) *P, R, F-M*
PMD			
Singleton	1, 1, 1	–	–
Factory method	1, 0.80, 0.88	0.93, 0.93, 0.93	–
Composite	0.88, 0.88, 0.88	0.88, 0.88, 0.88	–
Adapter	1, 1, 1	0.8, 1, 0.89	–
Decorator	1, 1, 1	–	0.77, 1, 0.87
Template method	1, 1, 1	0.67, 1, 0.80	–
Observer	1, 1, 1	0.92, 1, 0.93	0.76, 0.6, 0.67
Visitor	1, 0. 87, 0.93	1, 1, 1	0.73, 0.8, 0.76
Strategy	1, 0.97, 0.98	–	–
Command	1, 0.6, 0.75	–	–
Builder	0.92, 0.92, 0.92	0.9, 0.75, 0.81	–
Proxy	0.75, 1, 0.85	0.75, 1, 0.85	–
Iterator	0.8, 1, 0.88	0.80, 1, 0.89	–

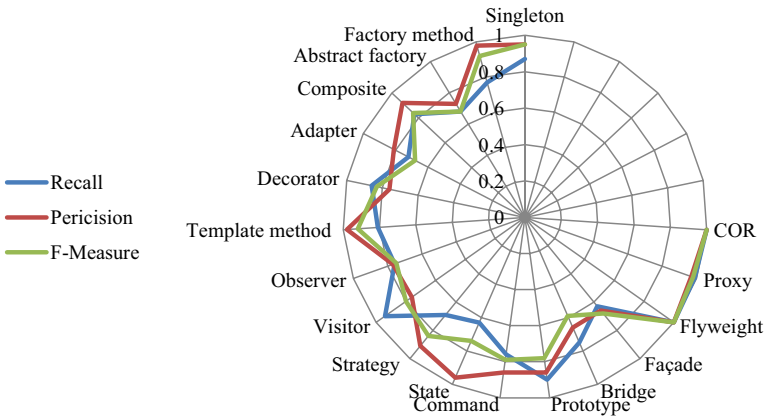

Fig. 30 Average *P, R* and *F-measure* for design patterns(JHotDrow)

that the techniques reducing the search space be chosen in such a way that they will not reduce accuracy.

The number of papers which can detect structural patterns from source code is limited to a particular language. Simply put, move towards the methods that are independent of languages is expected.

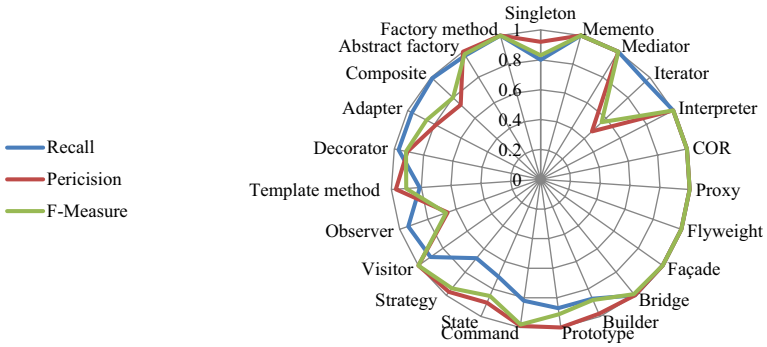

Fig. 31 Average *P, R* and *F-measure* for design patterns(JUnit)

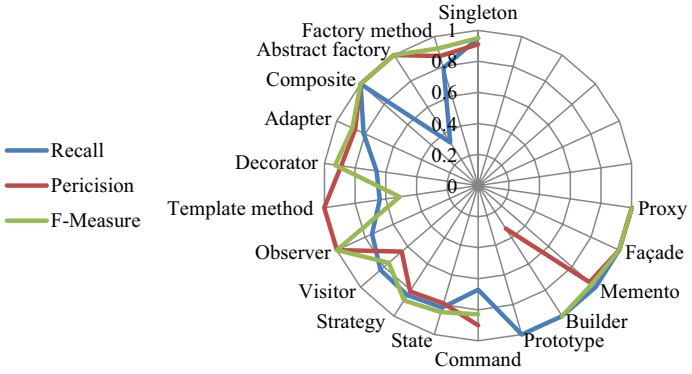

Fig. 32 Average P, R and F-measure for design patterns(JRefactory)

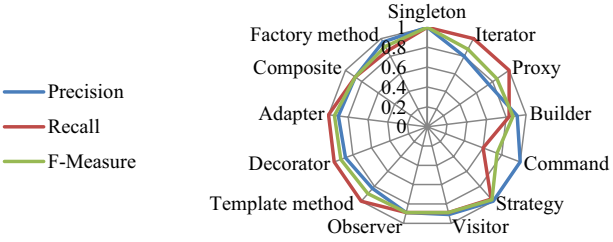

Fig. 33 Average *P, R* and *F-measure* for design patterns(PMD)

Another problem of these studies was that they emphasized only a few design patterns and a few categories. A number of papers, for example, De Lucia et al. (2009b) claim that they can detect behavioral patterns, but the effectiveness of their methods depends on the selection of test cases that cover behavior of the pattern. One reason for this issue is that the type of input selected for the method is not suitable, and the information which it provides does not meet the needs of the method for design pattern

discovery. This problem can be solved by the selection of different types of input. For example, some methods extract the sequence diagram from the source code and detect the design patterns based on the diagram, but it is likely that diagram does not correspond the source code and it will be noticed on the run-time. It is believed that there should be a move towards "Log Analyzer" and the use of the logs and other thing as the input. By considering the other types of input like "LOG", not only the methods will be able to detect behavioral design patterns but also their accuracy will improve. Dynamic analysis is important for detecting some design pattern and some papers, for example, Kirasić and Basch (2008) didn't support dynamic analysis. For behavioral patterns, precision is usually enhanced by obtaining dynamic information from monitoring activity of the executed program. This enables a detection mechanism to discriminate, for example, the Observer design pattern from the Bridge pattern (Alnusair et al. 2014) and we should move towards dynamic analysis. We expect the move towards the use of the tools that can extract exact diagrams from the source code (Oruc et al. 2016), but the detection process must be executed by means of log analysis during the run-time. Therefore, only patterns that occur at run-time will be detected and consideration of all patterns will be addressed.There may be a more important problem in calculation of the recall due to the absence of a sample in the problem tool. To reduce FP, another thing can be considered as an input. As mentioned in preceding section, source codes are considered as the input by different methods, but other things like comments can also be considered as input to achieve higher accuracy.

Another deficiency with the current literature is that most of the methods can only detect a group and a certain number of roles. For example, Chihada et al. (2015) proposed a method which can detect those patterns, which have certain number roles and it is applicable only to the patterns with four roles. The methods which are appropriate only for patterns with a certain number of roles return inexact and approximate solutions; they also generate many false positives. The approaches such as design pattern detection from source code using the classification approach, and comprehensive approach to the recover design pattern instances based on sub-patterns, and method signatures can detect design patterns that have a certain number of roles. However, methods based on graph theory can identify patterns with different roles. Although methods are encouraged to move towards using the graph theory.

Another problem is that many approaches cannot detect various implementations of design pattern and therefore, are dependent on specific implementations, but using machine learning, the various implementation of design patterns can be detected because it uses learning based on the information extracted from the source code to detect design pattern (Chihada et al. 2015), but learning based approaches have a in the sense that it is difficult to decide what part of the code should be selected for learning. This problem can be solved in future by considering the criteria and metrics for choosing part of the code.

Some methods which were proposed in papers cannot detect the design patterns having similar structures like state and strategy. This problem can be solved by behavioral analysis methods. Another problem that prevents the methods to detect the creational design patterns such as Singleton lies in the limitation of these methods to identify the relationships existing from a class to the design pattern, called self–loop. In Han et al. (2009), the Boolean function for the graph cannot be written and thus, this method cannot be used for these type of design patterns.

A number of methods are not fully automatic and man has a role in it, and this is not desired because FN and FP numbers may be affected due to human errors (Mayvan and

Table 15 The used datasets in Oruc et al. (2016)

Project	Number of classes
JUnit 3.8	54
JUnit 4.1	157
AWT 1.3	407

Rasoolzadegan 2017). Rasool et al. (2010) proposed the detection process which is semi-automatic so the source code should be annotated manually. However, this effect should be limited in future, and we expect to be fully automated in the future.

Query based approach is introduced. These approaches have a big problem to detect design pattern from source code. In this approach, detection of design patterns is performed by applying the query and information existing in the intermediate representation. However, the information that exist in the intermediate representation is not complete and there is no tool which can convert code to the intermediate representation completely (Mayvan and Rasoolzadegan 2017).

It has been showed in this paper that although design pattern detection has advanced a lot, there are still some subjects that can be improved. Many methods are focused on some design patterns, and most methods do not have a good performance. There are also weaknesses in their evaluation.

12 Conclusion

In this paper, all the aspects of design pattern detection have been examined, all the approaches, types of data representation, the methods, and many more. Design patterns have an important role in software engineering and help to understand codes. Design patterns are in software design phase to solve most of the problems in engineering and facilitate the production process and software development. But most of the systems have no document that engineers can recognize DPs from codes. Moreover, most of the time the implementation does not correspond to the design. Despite the fact that various design patterns were created and of great importance are design patterns in software engineering, the papers focused only on design patterns (GoF). It is noteworthy that some of them have not even been able to recognize all these 23 patterns, and only emphasized certain design patterns. Most papers could not detect design pattern in large cases, and most of them used special datasets. There are a lot of aspects that researcher can improve in future. A lot of things have been mentioned that researchers can use to develop design pattern detection. We believe that, if different methods are combined, many benefits can be gained.

Appendix

See Table 16.

Table 16 Summary of some papers about design pattern detection

No.	Paper	Approach	Category	Method	Advantage	Shortage	Evaluation result
1	Mayvan and Rasoolzadegan (2017)	graph-based	matching	Semantic matching +strong simulation algorithm	This method achieved around 100% precision and recall in most cases and the numbers of false positives were eliminated by using behavioral analysis	The number of FN and FP may be affected by human errors. This method is not automatic. The discovery of design pattern is based on the main body and their behavior. But finding an appropriate main body for one pattern is a challenge. In this paper, using tool, the source code was converted to UML diagram, but there is no guarantee that the diagram is complete and can yield the same result with large cases. This method is not language-independent and it's not known whether it works for different languages	In most cases, the proposed method achieved to P and R level around 100%; but the amount of F-measure was not reported, and for some of the patterns in some datasets, only the amount of P or R was reported

Table 16 (continued)

No.	Paper	Approach	Category	Method	Advantage	Shortage	Evaluation result
2	Dwivedi et al. (2016)	Learning-based	Supervised learning	In this paper, two classifiers including LRNN and random forest are considered. LRNN uses internal memory for executing random number of inputs, and random forest classifies input feature vectors as a predictive model	This method cannot detect all design patterns	Some methods have manual tagging techniques for preparation of dataset, but this task not only creates false positives but also is time consuming. In this method, object-oriented metrics for dataset preparation were considered to determine actual pattern instances	This paper reported F-measure for evaluation. Accuracy for LRNN is 97% and for random forest is 100%. Therefore random forest has a better performance compared to LRNN
3	García et al. (2013)	Reasoning-based	First order logic	Inference of first order logic with xml	This method evaluated on small test cases and can detect some design patterns	In this method, UML class and sequence diagrams are translated to ASP fact. ASP has some advantages over SPASS. For example ASP can handle larger input and more rules. It is also able to identify multiple and formalized possibilities	In this paper, no evaluation parameters are reported

Table 16 (continued)

No.	Paper	Approach	Category	Method	Advantage	Shortage	Evaluation result
4	Robinson and Bates (2016)	Parsing-based	Checking and parsing	This method uses CADP as formal verification. CADP is a model checker for verifying each pattern	This method is not appropriate for the large cases and the evaluation is performed on medium dataset	The model-checking phase reduces the number of false positives for command and strategy to zero	The effectiveness of method showing, the precision (30% on average) improved and this improvement have any effect of original recall
5	Issaoui et al. (2015)	Metric-based	Metric based filtering	This paper proposed a method that uses structural metrics to determine the most probable correspondences between the design elements and the design patterns	This method does not guarantee functionality with large systems	This method uses filtering to decrease complexity and to improve the limited performance	This method cannot detect state and strategy because they have similar structures, but it can detect singleton that most of methods could not detect. The evaluation of this method shows high precision and recall

Table 16 (continued)

No.	Paper	Approach	Category	Method	Advantage	Shortage	Evaluation result
6	Robinson and Bates (2016)	Similarity scoring	Matching	The proposed method calculates normalized cross correlation between system and pattern matrix. The degree of similarity between them was show by normalized cross correlation	This method calculates the normalized cross correlation between system matrix and pattern matrix and calculates similarity between two sub-graph rather than vertices	This method calculates the normalized cross correlation between system matrix and pattern matrix. It calculates similarity between two sub-graph rather than vertices. This paper cannot distinguish between behavioral patterns. Although this method encodes eight design features, it cannot be claimed that it is complete	This method not only can find exact matches of pattern instances, but also can identify their possible variants
7	Bouassida and Ben-Abdallah (2010b)	XML document retrieval	XML query	This method uses context similarity function to determine correspondences between classes of the design and pattern	This method can use for each structure and method correspondences and this method can tolerate certain structural differences in the design compared to the pattern.	There is no report for evaluation parameters like P, R and F-m	It has no evaluation

Table 16 (continued)

No.	Paper	Approach	Category	Method	Advantage	Shortage	Evaluation result
8	Kniesel and Binun (2009a)	Data fusion	Analyzing and result combining	This approach analyzes the aspects of patterns by tools then there is combining phase. In this phase the expertise of tools is combined Finally contributes of analysis estimates of which patterns are likely to be relevant	This method can detect design pattern from the inputs which are not complete and detects the design patterns which individual tools cannot detect	There is no report for evaluation parameters like P, R and F-m	For the decorator, visitor and observer pattern, the witness-based approach yields better precision and recall than provided by any single tool and on the analyzed instances of the Bridge, Mediator and Facade pattern, data fusion could not improve results

References

Aladib L, Lee S (2018) Pattern detection and design rationale traceability: an integrated approach to software design quality. IET Softw 13:249–259

Alhusain S, Coupland S, John R, Kavanagh M (2013a) Towards machine learning based design pattern recognition. In: 2013 13th UK workshop on computational intelligence (UKCI). IEEE, pp 244–251

Alhusain S, Coupland S, John R, Kavanagh M (2013b) Design pattern recognition by using adaptive neuro fuzzy inference system. In: 2013 IEEE 25th international conference on tools with artificial intelligence (ICTAI). IEEE, pp 581–587

Alnusair A, Zhao T (2009) Towards a model-driven approach for reverse engineering design patterns. In Proceedings of the 2nd international workshop on transforming and weaving ontologies in MDE (TWOMDE 2009), Denver, Colorado, USA, vol. 531, p 130

Alnusair A, Zhao T (2010) Using semantic inference for software understanding and design recovery. In: 2010 seventh international conference on information technology: new generations (ITNG). IEEE, pp 980–985

Alnusair A, Zhao T, Yan G (2013) Automatic recognition of design motifs using semantic conditions. In: Proceedings of the 28th annual acm symposium on applied computing. ACM, pp 1062–1067

Alnusair A, Zhao T, Yan G (2014) Rule-based detection of design patterns in program code. Int J Softw Tools Technol Transf 16(3):315–334

Al-Obeidallah M, Petridis M, Kapetanakis S (2016) A survey on design pattern detection. Int J Softw Eng (IJSE) 7(3):41–59

Al-Obeidallah MG, Petridis M, Kapetanakis S (2017a) A structural rule-based approach for design patterns recovery. In international conference on software engineering research, management and applications. Springer, Cham, pp 107–124

Al-Obeidallah M, Petridis M, Kapetanakis S (2017b) MLDA: a multiple levels detection approach for design patterns recovery. In: Proceedings of the international conference on compute and data analysis. ACM, pp 33–40

Alshira'h M (2017) Detection a design pattern through merge static and dynamic analysis using altova and lambdes tools. Int J Appl Eng Res 12(19):8518–8522

Arcelli F, Tosi C, Zanoni M, Maggioni S (2008) The MARPLE project: a tool for design pattern detection and software architecture reconstruction. In: 1st international workshop on academic software development tools and techniques (WASDeTT-1), pp 325–334

Ba-Brahem AS, Qureshi M (2014) The proposal of improved inexact isomorphic graph algorithm to detect design patterns. arXiv preprint arXiv:1408.6147

Bashir A, Rasool G, Bashir K, Ali AH (2013) Design patterns and documentation recovery based on attributes. Int J Soft Comput Softw Eng 3(3):149–155

Bayley I, Zhu H (2010) Formal specification of the variants and behavioural features of design patterns. J Syst Softw 83(2):209–221

Bernardi, ML, Di Lucca GA (2010) Model-driven detection of design patterns. In: 2010 IEEE international conference on software maintenance (ICSM). IEEE, pp 1–5

Bernardi ML, Cimitile M, Di Lucca GA (2013) A model-driven graph-matching approach for design pattern detection. In: 2013 20th working conference on reverse engineering (WCRE). IEEE, pp 172–181

Bernardi ML, Cimitile M, Di Lucca G (2014) Design pattern detection using a DSL-driven graph matching approach. J Softw Evol Process 26(12):1233–1266

Bernardi ML, Cimitile M, De Ruvo G, Di Lucca GA, Santone A (2015) Improving design patterns finder precision using a model checking approach. In: CAiSE forum, pp 113–120

Binun A, Kniesel G (2012a) DPJF-design pattern detection with high accuracy. In: 2012 16th European conference on software maintenance and reengineering (CSMR). IEEE, pp 245–254

Binun A, Kniesel G (2012b) Joining forces for higher precision and recall of design pattern detection. CS Department III, University of Bonn, Germany, Technical report IAI-TR-2012-01

Bouassida N, Ben-Abdallah H (2009) Design improvement through dynamic and Structural pattern identification. In: Proceedings of the third international conference on innovation and information and communication technology. BCS Learning & Development Ltd., pp 4–4

Bouassida N, Ben-Abdallah H (2010a) A new approach for pattern problem detection. In: International conference on advanced information systems engineering. Springer, Berlin, pp 150–164

Bouassida N, Ben-Abdallah H (2010b) Pattern and spoiled pattern detection through an information retrieval approach. J Emerg Technol Web Intell 2(3):167

Chaturvedi A, Gupta M, Gupta SK (2016) Design pattern detection using genetic algorithm for sub-graph isomorphism to enhance software reusability. Int J Comput Appl 135(4):33–36

Chaturvedi A, Gupta M, Gupta SK (2018) DPVO: design pattern detection using vertex ordering a case study in JHotDraw with documentation to improve reusability. In: International conference on communication, networks and computing. Springer, Singapore, pp 452–465
Chen L, Qiu M (2010) An algorithm for automatic mining design pattern. In: 2010 5th international conference on computer science and education (ICCSE). IEEE, pp 1860–1864
Chihada A, Jalili S, Hasheminejad SMH, Zangooei MH (2015) Source code and design conformance, design pattern detection from source code by classification approach. Appl Soft Comput 26:357–367
Czibula IG, Czibula G (2008) Identifying design patterns in object-oriented software systems using unsupervised learning. In: IEEE International conference on automation, quality and testing, robotics, 2008, AQTR 2008, vol. 3. IEEE, pp 347–352
Dabain H, Manzer A, Tzerpos V (2015) Design pattern detection using FINDER. In: Proceedings of the 30th annual acm symposium on applied computing. ACM, pp 1586–1593
De Lucia A, Deufemia V, Gravino C, Risi M (2009a) Design pattern recovery through visual language parsing and source code analysis. J Syst Softw 82(7):1177–1193
De Lucia A, Deufemia V, Gravino C, Risi M (2009b) Behavioral pattern identification through visual language parsing and code instrumentation. In: 13th European conference on software maintenance and reengineering 2009 CSMR'09. IEEE, pp 99–108
De Lucia A, Deufemia V, Gravino C, Risi M (2010a) Improving behavioral design pattern detection through model checking. In: 2010 14th European conference on software maintenance and reengineering. IEEE, pp 176–185
De Lucia A, Deufemia V, Gravino C, Risi M (2010b) An eclipse plug-in for the detection of design pattern instances through static and dynamic analysis. In: 2010 IEEE international conference on software maintenance (ICSM). IEEE, pp 1–6
De Lucia A, Deufemia V, Gravino C, Risi M, Pirolli C (2015a) ePadEvo: a tool for the detection of behavioral design patterns. In: 2015 IEEE international conference on software maintenance and evolution (ICSME). IEEE, pp 327–329
De Lucia A, Deufemia V, Gravino C, Risi M (2015b) Towards automating dynamic analysis for behavioral design pattern detection. In: 2015 IEEE international conference on software maintenance and evolution (ICSME). IEEE, pp 161–170
Di Martino B, Esposito A (2013) Automatic recognition of design patterns from UML-based software documentation. In: Proceedings of international conference on information integration and web-based applications and services. ACM
Di Martino B, Esposito A (2016) A rule-based procedure for automatic recognition of design patterns in UML diagrams. Softw Pract Exp 46(7):983–1007
Dong J, Lad DS, Zhao Y (2007) DP-Miner: design pattern discovery using matrix. In: 14th Annual IEEE international conference and workshops on engineering of computer-based systems, 2007, ECBS'07. IEEE, pp 371–380
Dong J, Sun Y, Zhao Y (2008a) Compound record clustering algorithm for design pattern detection by decision tree learning. In: IEEE international conference on information reuse and integration, IRI 2008. IEEE, pp 226–231
Dong J, Sun Y, Zhao Y (2008b) Design pattern detection by template matching. In: Proceedings of the 2008 ACM symposium on applied computing. ACM, pp 765–769
Dong J, Zhao Y, Peng T (2009a) A review of design pattern mining techniques. Int J Softw Eng Knowl Eng 19(06):823–855
Dong J, Zhao Y, Sun Y (2009b) A matrix-based approach to recovering design patterns. IEEE Trans Syst Man Cybern Part A Syst Hum 39(6):1271–1282
Dwivedi AK, Tirkey A, Rath SK (2016) Applying software metrics for the mining of design pattern. In: 2016 IEEE Uttar Pradesh section international conference on electrical, computer and electronics engineering (UPCON). IEEE, pp 426–431
Dwivedi AK, Tirkey A, Ray RB, Rath SK (2016) Software design pattern recognition using machine learning techniques. In Region 10 conference (TENCON), 2016 IEEE. IEEE, pp 222–227
Dwivedi AK, Tirkey A, Rath SK (2018) Software design pattern mining using classification-based techniques. Front Comput Sci 12(5):908–922
Elaasar M, Briand LC, Labiche Y (2015) VPML: an approach to detect design patterns of MOF-based modeling languages. Softw Syst Model 14(2):735–764
Fontana FA, Zanoni M (2011) A tool for design pattern detection and software architecture reconstruction. Inf Sci 181(7):1306–1324
Fontana FA, Maggioni S, Raibulet C (2011) Understanding the relevance of micro-structures for design patterns detection. J Syst Softw 84(12):2334–2347

Freeman E, Robson E, Bates B, Sierra K (2004) head first design patterns: a brain-friendly guide. O'Reilly Media Inc., Sebastopol
Gamma E (1995) Design patterns: elements of reusable object-oriented software. Pearson Education India, New York
García A, Velasco-Elizondo P, Zamarrón JM (2013) Applying social network analysis metrics to software design patterns detection. In: ASNA 2013 applications of social network analysis, University of Zurich
Gautam AK, Diwaker S (2012) Automatic detection of software design patterns from reverse engineering. Springer, Berlin
Guéhéneuc YG (2007) P-mart: pattern-like micro architecture repository. In: Proceedings of the 1st EuroP-LoP focus group on pattern repositories, pp 1–3
Guéhéneuc YG, Antoniol G (2008) Demima: a multilayered approach for design pattern identification. IEEE Trans Softw Eng 34(5):667–684
Guéhéneuc YG, Guyomarćh JY, Sahraoui H (2010) Improving design-pattern identification: a new approach and an exploratory study. Softw Qual J 18(1):145–174
Gupta M (2011) Design pattern mining using greedy algorithm for multi-labelled graphs. Int J Inf Commun Technol 3(4):314–323
Gupta M (2017) A technique for design patterns detection. Indian J Comput Sci Eng (IJCSE) 8(3)
Gupta M, Rao RS (2014) Design pattern mining by product of sum (POS) expression for graphs. Int J Comput Appl. https://doi.org/10.5120/14856-3223
Gupta M, Pande A, Rao RS, Tripathi AK (2010a) Design pattern detection by normalized cross correlation. In: 2010 international conference on methods and models in computer science (ICM2CS). IEEE, pp 81–84
Gupta M, Rao RS, Tripathi AK (2010b) Design pattern detection using inexact graph matching. In: 2010 international conference on communication and computational intelligence (INCOCCI). IEEE, pp 211–217
Gupta M, Pande A, Tripathi AK (2011b) Design patterns detection using SOP expressions for graphs. ACM SIGSOFT Softw Eng Notes 36(1):1–5
Han Z, Wang L, Yu L, Chen X, Zhao J, Li X (2009) Design pattern directed clustering for understanding open source code. In: IEEE 17th international conference on program comprehension, 2009, ICPC'09. IEEE, pp 295–296
Hayashi S, Katada J, Sakamoto R, Kobayashi T, Saeki M (2008) Design pattern detection by using meta patterns. IEICE Trans Inf Syst 91(4):933–944
Hussain S, Keung J, Khan AA (2017) Software design patterns classification and selection using text categorization approach. Appl Soft Comput 58:225–244
Hussain S, Keung J, Sohail MK, Khan AA, Ilahi M (2019) Automated framework for classification and selection of software design patterns. Appl Soft Comput 75:1–20
Issaoui I, Bouassida N, Ben-Abdallah H (2015) Using metric-based filtering to improve design pattern detection approaches. Innov Syst Softw Eng 11(1):39–53
Issaoui I, Bouassida N, Ben-Abdallah H (2016) Predicting the existence of design patterns based on semantics and metrics. Int Arab J Inf Technol 13(2):310–319
Kaczor O, Guéhéneuc YG, Hamel S (2010) Identification of design motifs with pattern matching algorithms. Inf Softw Technol 52(2):152–168
Karam M, Aljahdali S, Mcheick H, Abdallah R, Ollaic H (2014) Graph transformation rules with weight and fuzzy logic for better design pattern recognition. Int J Comput Inf Technol 3
Kirasić D, Basch D (2008) Ontology-based design pattern recognition. In: International conference on knowledge-based and intelligent information and engineering systems. Springer, Berlin, pp 384–393
Kitchenham B, Charters S (2007) Guidelines for performing systematic literature reviews in software engineering
Kniesel G, Binun A (2009a) Witnessing patterns: a data fusion approach to design pattern detection. CS Department III, University of Bonn, Germany, Technical report IAI-TR-2009-02, ISSN, 0944-8535
Kniesel G, Binun A (2009b) Standing on the shoulders of giants---a data fusion approach to design pattern detection. In: IEEE 17th international conference on program comprehension, 2009, ICPC'09. IEEE, pp 208–217
Lavender RG, Schmidt DC (1995) Active object—an object behavioral pattern for concurrent programming
Lebon M, Tzerpos V (2012) Fine-grained design pattern detection. In: 2012 IEEE 36th annual computer software and applications conference (COMPSAC). IEEE, pp 267–272
Lee H, Youn H, Lee E (2008) A design pattern detection technique that aids reverse engineering. Int J Secur Appl 2(1):1–12

Liamwiset C, Wiwat V (2013) Detection of design patterns in software design model using graph. In: Applied mechanics and materials, vol. 411. Trans Tech Publications, pp 559–562
Liu C, van Dongen BF, Assy N, van der Aalst WM (2018a) A framework to support behavioral design pattern detection from software execution data. In: ENASE, pp 65–76
Liu C, van Dongen BF, Assy N, van der Aalst WM (2018b) A framework to support behavioral design pattern detection from software execution data. In: ENASE, pp 65–76
Luitel G, Stephan M, Inclezan D (2016) Model level design pattern instance detection using answer set programming. In: Proceedings of the 8th international workshop on modeling in software engineering. ACM, pp 13–19
Majtás LU (2011) Contribution to the creation and recognition of the design patterns instances. Inf Sci Technol Bull ACM Slovak 3(1):84–92
Mattson TG, Sanders B, Massingill B (2004) Patterns for parallel programming. Pearson Education, New York
Mayvan BB, Rasoolzadegan A (2017) Design pattern detection based on the graph theory. Knowl Based Syst 120:211–225
Nagy A, Kovari B (2015) Programming language neutral design pattern detection. In: 2015 16th IEEE international symposium on computational intelligence and informatics (CINTI). IEEE, pp 215–219
Ng JKY, Guéhéneuc YG, Antoniol G (2010) Identification of behavioural and creational design motifs through dynamic analysis. J Softw Evol Process 22(8):597–627
Oruc M, Akal F, Sever H (2016) Detecting design patterns in object-oriented design models by using a graph mining approach. In: 2016 4th international conference on software engineering research and innovation (CONISOFT). IEEE, pp 115–121
Pande A, Gupta M, Tripathi AK (2010a) A new approach for detecting design patterns by graph decomposition and graph isomorphism. Contemporary computing, pp 108–119
Pande A, Gupta M, Tripathi AK (2010b) DNIT—a new approach for design pattern detection. In: 2010 international conference on computer and communication technology (ICCCT). IEEE, pp 545–550
Pande A, Gupta M, Tripathi AK (2010c) A decision tree approach for design patterns detection by subgraph isomorphism. In: International conference on advances in information and communication technologies. Springer, Berlin, pp 561–564
Panich A, Vatanawood W (2016) Detection of design patterns from class diagram and sequence diagrams using ontology. In: 2016 IEEE/ACIS 15th international conference on computer and information science (ICIS). IEEE, pp 1–6
Paydar S, Kahani M (2012) A semantic web based approach for design pattern detection from source code. In: 2012 2nd international econference on computer and knowledge engineering (ICCKE). IEEE, pp 289–294
Pradhan P, Dwivedi AK, Rath SK (2015) Detection of design pattern using graph isomorphism and normalized cross correlation. In: 2015 Eighth international conference on contemporary computing (IC3). IEEE, pp 208–213
Priya RK (2014) A survey: design pattern detection approaches with metrics. In: 2014 IEEE national conference on emerging trends in new and renewable energy sources and energy management (NCET NRES EM). IEEE, pp 22–26
Qiu M, Jiang Q, Gao A, Chen E, Qiu D, Chai S (2010) Detecting design pattern using subgraph discovery. In: Asian conference on intelligent information and database systems. Springer, Berlin, pp 350–359
Rasool G, Mäder P (2011) Flexible design pattern detection based on feature types. In: 2011 26th IEEE/ACM international conference on automated software engineering (ASE). IEEE, pp 243–252
Rasool G, Mäder P (2014) A customizable approach to design patterns recognition based on feature types. Arab J Sci Eng 39(12):8851–8873
Rasool G, Philippow I, Mäder P (2010) Design pattern recovery based on annotations. Adv Eng Softw 41(4):519–526
Ren W, Zhao W (2012) An observer design-pattern detection technique. In: 2012 IEEE international conference on computer science and automation engineering (CSAE), vol. 3. IEEE
Riaz M, Breaux T, Williams L (2015) How have we evaluated software pattern application? A systematic mapping study of research design practices. Inf Softw Technol 65:14–38
Robinson A, Bates C (2016) Recovering design patterns from large codebases. In: Proceedings of international conference on computer science education innovation & technology (CSEIT). Global Science and Technology Forum, p 136
Romano S, Scanniello G, Risi M, Gravino C (2011) Clustering and lexical information support for the recovery of design pattern in source code. In: 2011 27th IEEE international conference on software maintenance (ICSM). IEEE, pp 500–503

Sebastian F (2002) Machine learning in automated text categorization. J ACM Comput Surv (CSUR) 34:1–47
Stencel K, Wegrzynowicz P (2008) Detection of diverse design pattern variants. In: Software engineering conference, 2008, APSEC'08, 15th Asia-Pacific. IEEE, pp 25–32
Stephan M, Cordy JR (2015) Identifying instances of model design patterns and antipatterns using model clone detection. In: 2015 IEEE/ACM 7th international workshop on modeling in software engineering (MiSE). IEEE, pp 48–53
Stoianov A, Şora I (2010) Detecting patterns and antipatterns in software using prolog rules. In: 2010 International joint conference on computational cybernetics and technical informatics (ICCC-CONTI). IEEE, pp 253–258
Thaller H, Linsbauer L, Egyed A (2019) Feature maps: a comprehensible software representation for design pattern detection. In: 2019 IEEE 26th international conference on software analysis, evolution and reengineering (SANER), pp 207–21
Thankappan J, Patil V (2015) Detection of web design patterns using reverse engineering. In: 2015 Second international conference on advances in computing and communication engineering (ICACCE). IEEE, pp 697–701
Thongrak M, Vatanawood W (2014) Detection of design pattern in class diagram using ontology. In: Computer science and engineering conference (ICSEC). IEEE, pp 97–102
Tramontana E (2014) Detecting extra relationships for design patterns roles. In: Proceedings of AsianPlop, Tokyo, Japan
Tripathi V, Mahesh TSG, Srivastava A (2009) Performance and language compatibility in software pattern detection. In 2009 IEEE international conference on advance computing, IACC 2009. IEEE, pp 1639–1643
Tsantalis N, Chatzigeorgiou A, Stephanides G, Halkidis ST (2006) Design pattern detection using similarity scoring. IEEE Trans Softw Eng 32(11):896–909
Uchiyama S, Washizaki H, Fukazawa Y, Kubo A (2011) Design pattern detection using software metrics and machine learning. In: First international workshop on model-driven software migration (MDSM 2011), p 38
Uchiyama S, Kubo A, Washizaki H, Fukazawa Y (2014) Detecting design patterns in object-oriented program source code by using metrics and machine learning. J Softw Eng Appl 7(12):983
von Detten M (2011) Towards systematic, comprehensive trace generation for behavioral pattern detection through symbolic execution. In: Proceedings of the 10th ACM SIGPLAN-SIGSOFT workshop on program analysis for software tools. ACM, pp 17–20
von Detten M, Becker S (2011) Combining clustering and pattern detection for the reengineering of component-based software systems. In: Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on quality of software architectures–QoSA and architecting critical systems–ISARCS. ACM, pp 23–32
von Detten M, Platenius MC (2009) Improving dynamic design pattern detection in reclipse with set objects. In: Proceedings of the 7th international Fujaba days, pp 15–19
Wang L, Han Z, He J, Wang H, Li X (2012a) Recovering design patterns to support program comprehension. In: Proceedings of the 2nd international workshop on evidential assessment of software technologies. ACM, pp 49–54
Wang Y, Guo H, Liu H, Abraham A (2012b) A fuzzy matching approach for design pattern mining. J Intell Fuzzy Syst 23(2–3):53–60
Wegrzynowicz P, Stencel K (2013) Relaxing queries to detect variants of design patterns. In: 2013 federated conference on computer science and information systems (FedCSIS). IEEE, pp 1571–1578
Wierda A, Dortmans E, Somers L (2008) Pattern detection in object-oriented source code. In: Software and data technologies. Springer, Berlin, pp 141–158
Yu D, Zhang Y, Ge J, Wu W (2013a) From sub-patterns to patterns: an approach to the detection of structural design pattern instances by subgraph mining and merging. In: 2013 IEEE 37th annual computer software and applications conference (COMPSAC). IEEE, pp 579–588
Yu D, Ge J, Wu W (2013b) Detection of design pattern instances based on graph isomorphism. In: 2013 4th IEEE international conference on software engineering and service science (ICSESS). IEEE, pp 874–877
Yu D, Zhang Y, Chen Z (2015) A comprehensive approach to the recovery of design pattern instances based on sub-patterns and method signatures. J Syst Softw 103:1–16
Yu D, Zhang P, Yang J, Chen Z, Liu C, Chen J (2018) Efficiently detecting structural design pattern instances based on ordered sequences. J Syst Softw 142:35–56
Zanoni M, Fontana FA, Stella F (2015) On applying machine learning techniques for design pattern detection. J Syst Softw 103:102–117

Zhang P, Yu D, Wang J (2017) A degree-driven approach to design pattern mining based on graph matching. In: 2017 24th Asia-Pacific software engineering conference (APSEC). IEEE, pp 179–188

Zhu H, Bayley I, Shan L, Amphlett R (2009) Tool support for design pattern recognition at model level. In: 2009 33rd Annual IEEE International Computer Software and Applications Conference, 2009, COMPSAC'09, vol. 1. IEEE, pp. 228–233