IJCSE International Journal of Computer Sciences and Engineering Open Access

Survey Paper Vol.-6, Issue-8, Aug 2018 E-ISSN: 2347-2693

Present Approaches for Detection of Design Pattern: A Survey

A. Chaturvedi

School of studies in Computer Science and Applications, Jiwaji University, Gwalior (M.P.), India

Corresponding Author :arti.2408@gmail.com Tel.: 9926485989

Available online at: www.ijcseonline.org

Accepted: 13/Aug/2018, Published: 31/Aug/2018

Abstract- Existing software's are implemented by a third party and open source software may take a lot of time to understand, and patterns are applied without explicit class name, comments, or attached documents. If better reusability is required for an existing application where design patterns were used, then an approach that can detect the used design pattern in the existing application will be useful. Therefore, a reliable design pattern detection approach is required to promote software reusability. Design pattern detection is expected to improve the understandability and reusability of existing software. This paper represents the background work of design pattern detection. I review different approaches that have been documented so far in the literature and present the tools that have been developed. Pattern detection approaches are classified into structural analysis, behavioral analysis, and semantic analysis to mining the design pattern from the source code of different legacy application. Structural analysis approaches based on recovering the structural relationship from different artifacts available in the source code. Behavioral analysis approaches take in account the execution behavior of the program and this analysis is dynamic which execute run time behavior of the software. Semantic analysis approaches are combination of both, structure and behavioral analysis for verifying the accuracy of found result. In this paper I propose a survey of structural analysis approaches for design pattern detection.

Keywords- Design Pattern, UML, Ontology, Sub-Graph Isomorphism, Structural Analysis.

I. INTRODUCTION

Object oriented design pattern is a general repeatable solution to a commonly occurring problem in software development, and are considered as standard of "good" software designs. The idea of patterns was firstly introduced by Christopher Alexander, in the field of architecture. In future the concept of patterns has been changed in order to fit software design by Gamma, Helm, Johnson and Vlissides [48]. The authors catalogued 23 design patterns, known as GoF design patterns. The Concept of pattern is widely used in software development to facilitate the software reuse. Moreover, a design pattern can be reused as a building block for better software implementation and their documentation in a software system can improve software reusability and program understanding. In OO software development, object and classes is fundamental reusable unit but they alone would not be enough for an effective reuse. Therefore OO methodologies gave birth of design pattern for developing OO application [49]. The purpose for using design pattern in software development is to improve the reusability and the quality of software .The motivation for formalizing design pattern is to make them easier to understand and implement in new software application [1].

Design pattern provide ways to structure software modules into system that are flexible, extensible, and have a high degree of reusability. Design pattern are an attempt to capture expertise in building object oriented software that describes solution to a repeated design problem in a logical and general way. Gamma at el [48] defines design pattern as "description of communicating objects and classes that are customized to solve a general design problem in a particular context." A design pattern names, abstracts, and identifies the basic features of a common design structure that makes it beneficial for creating a reusable object –oriented design. The design pattern recognizes the participating classes and their instances, their roles and collaboration, and the distribution of responsibilities. Object-oriented design pattern usually show relationships and interaction between classes or objects, without specifying the final application classes or objects that are involved. [50]

The identification of design patterns as part of the reengineering process can convey important information to the designers. [33] "The central objective of pattern identification approaches is to accurately detect patterns from the source code which improve the software reusability, maintenance, program comprehension, refactoring, restructuring, reverse engineering and reengineering disciplines" . [51] "Design pattern is useful to

gain knowledge on the design issues of an existing system, on its architecture and design quality, improving the comprehension of the system and hence its reusability, maintainability and evolution" [52].
The rest of the paper is organized as follows: review on different approaches is proposed in section II. Section III explores review on graph based approaches. And conclusion is given in section IV.

II. DIFFERENT APPROACHES AND TOOLS FOR DESIGN PATTERN DETECTION

In this section, I review the various existing approaches that discover the design pattern.

In paper[2] , authors are introduce a detection approach for Creational design pattern such that "Abstract Factory", "Factory Method", "Builder", "Prototype" and "Singleton". Now this approach is based on anti-patterns characteristic. According to author's "anti-patterns are bad alternative solutions to the design pattern, those are termed as missing design patterns". This approach performs in three levels: "structural", "behavioral", and "semantic" analysis and every level analyze anti-patterns information of particular design pattern in existing software design. The result of this analysis is presents in a tool named as "Anti-pattern" based "Creational Design Pattern Recommendation (ACDPR)". After detection of anti-pattern information of particular design pattern, this tool gives a score to such design pattern and this score determine the approval of design pattern. This tool is implemented in Java and 21 existing software were used as dataset.

In paper [3], authors are proposed machine learning based approach for design pattern detection. This approach has two stages for design pattern detection. First stage for prepare dataset which complete in following steps: 1) Define design pattern template element such as problem, solution, applicability etc. 2) Select the patterns participants such that number of classes in a pattern, role of each class etc. 3) Prepare Object Oriented metrics vector and using various pattern detection tools. Here this approach consider 67 different number of Object Oriented metrics types. Also store metrics value of all patterns participants in a single row .If one pattern has four numbers of participants then it includes 67x4=268 number of features vector for single instance of that pattern. Second stage also complete in three sub process: 1) Here two classifiers such as "Layer Recurrent Neural Network" and "Decision Tree" used for learning process. These classifiers validate features of pattern participants in software i.e. check either pattern participants present in software or not. 2) In next sub process, classifiers are removing unfitted features of pattern participants. 3) And last verify the results.

Reference [4] Proposed a design pattern identification approach based on similarity between design pattern and domain matrixes. This is two phase approach. In first phase, this method finds matching between existing software and design pattern using matrix where rows are shows keywords of the pattern and columns represents in term of design (like class name of design) . After obtaining different matrices, they are listing the matrices which have maximum score for measured the function. In second phase, this approach applies some question on matrices which is found in first phase. These questions are explaining design of specific design pattern. This phase can be repeated until the specific design pattern will be identified.

In paper [5], authors are proposed a pattern identification algorithm "*SiDiff"* which measures difference between two graph structure diagrams, one is design pattern graph and other is existing software design graph. This algorithm compares two graphical figure components and calculates the similarity of pair figure component. If two components of both graphs are found to be similar to each other but not similar to other components then they are matched. In this algorithm each class of UML diagram has fixed criteria and their weight value. The similarity verification is based on these criteria weight value. This approach also detects incomplete instances of design pattern. In this paper authors are not produce experimental results for any live software.
In paper [6], authors are proposed a design pattern detection technique based on some fixed attributes of a specific design pattern. These attributes are "structural", "relational" and "behavioral". This process first define pattern attribute such that name of participating classes, "Generalization" and "Aggregation" relationship between two classes, "Method return type" etc. This process also focuses on different implementation variation of particular design pattern. Moreover, next this process searches these attributes in existing software using different detection approaches. In this paper attributes of "Factory Method" design pattern and "Strategy" design pattern are define and as domain this approach uses "JHotDraw", "JRefactory" and "JUnit" software. These researchers also proposed implementation variants of "*Abstract Factory", "Decorator", "Adapter", "Proxy", "Chain of Responsibility", and "Façade"* design patterns in [7] and [8].

In paper [9] authors are proposed a machine learning based technique for design pattern detection. They uses "Columbus framework" as domain where source code of existing software is converted into ASG ("Abstract Syntax Graph"). Here, the authors are applying some predictors (such as whether a participant class has a base or not, or how many new method define by participant class) on structure instance of specific design pattern which is recovered from Columbus framework. Moreover, ASG calculate predictor value and manually inspect source code of software for verification of that design pattern. Afterwards they propose a machine learning system with predictor value. This system based on “decision tree” and “back propagation neural network”. Finally the outputs of machine learning system merge with “Columbus framework”. In this paper authors are defining predictor for “Adapter” and “Strategy” design pattern.

In paper [10] authors are using a “Template matching” technique for design pattern detection. This method calculates “normalized cross correlation” between sub part of existing software matrix and design pattern matrix. “Normalized cross correlation” measures similarity degree between specific design pattern and sub part of existing software. If a sub part of existing software is similar to the design pattern than an instance of that design pattern is found. In this technique, not only the exact match of pattern instance is detected from existing software source code, but also the pattern variation can be identified.

In [11] authors are proposed “Micro Structure” as software module based on object oriented design. This process examines the role of participating classes of that structure manually. In this process authors are create a class library and calculates metrics value for each class such that “Size of class” (in terms of number of methods and fields), “Filiation of class” (in terms of number of parents class, number of child class), “Cohesion value of class” (in terms of degree of both, method and attributes belong together), and “Coupling between classes” (in terms of collaboration of one class to another class). Moreover, authors are applying a “rule learner” algorithm with set of metrics value and compare these values with metrics value of library classes using “leave-one-out” method. This approach considers six open source software’s for find out pattern instances and finally they create a pattern repository.

In paper [12], authors are present a novel technique for design pattern detection that uses machine learning technique and calculate software matrices. This method is completed in two phases. First is “**learning phase”** ,where authors define five patterns of GoF, such as “Singleton”, “Template Method”, “Adapter”, “State”, and “Strategy” and consider 12 roles (patterns class name) of these patterns such as for “singleton” (“singleton”), for “Template Method” (“Abstract class”, “Concrete class”), for “Adapter” (“Target”, “Adapter”, “Adaptee”) etc. Now this process decide matrices using “Goal Question Metrics” (“GQM”) such as to abstract class role identification the GQM is “are abstract method defined?” etc. and then they apply neural network algorithm and find out the value of each role . Second phase is “Detection phase**”**. Moreover, the output of learning phase is input of detection phase, so in this phase, determine role candidate using machine learning simulator and input these role candidate in pattern detection system according to definition of pattern structure and get an occurrence of specific design pattern.

In paper [13], authors are presents a method for reorganization of design pattern in source code through dynamic and static analysis method combination. The implementation procedure has three different steps: 1) “Static Parser”. 2) “Dynamic validation” and 3) “SWRL rules”. The source code is examined through AST (“abstract syntax tree”) parser. Meanwhile, rules set is in certain design pattern manually describe in “SWRL”. These “SWRL rules” are defined in “OWL” format, and then “OWL” individuals map with ontology model. Finally dynamic study is working to deteriorate or approve the candidate outcome. Another “ontology-based” three layer approach for design pattern reorganization is proposed by Damir Kirasic [14]. Here, in first layer, existing software source code is converted into “AST” and “XML” form. Second layer create structural features of design pattern and rules of programming concepts in “OWL” ontology form and finally a tool “analyzer” identify design pattern in “XML” of existing software source code. For parsing source code, this process using “ANTLR” framework. In paper [15], also proposed method is based on “ontology” idea. This process explain class structure of specific design pattern in “Semantic Query-enhances Web Rules Language” (“SQWRL”) and Class diagram of source code define in UML and XMI format. Moreover, a tool (developed by them) “XSLT” is converted this “XMI” into “OWL”(“Web Ontology Language”). This process identifies specific design pattern represent by “SQWRL” in “OWL” (source code) and identifies exact matching of design pattern and their location in domain (existing software).

In [16], MARPLE (“Metrics and Architecture Reconstruction Plug-in for Eclipse”) is a design pattern recognition tool which determines candidate design pattern instances by applying data mining approach using “structural query”. This method creates “elemental design pattern” which is similar to structure of design pattern or sub pattern. Moreover, in this system existing software source code represents in AST and then extract structured facts applying simple “structural query”. These facts are store in XML file. Moreover this XML file analyses by a sub-system “Joiner” which define rules for design patterns. “Joiner” supports design pattern in graph structure where classes of existing software represented by “nodes” and “edges” define structure of “design pattern or sub-patterns”. Moreover, by Appling a “graph matching query”, this “joiner” finds out design pattern instances. Another sub system of “MARPLE” is “Classifier” which verifies that output of “joiner” is actual design pattern instance or not. In this work, five design patterns proposed as an output

"Singleton", "Adapter", "Composite", "Decorator", Factory Method".

In [17], authors are suggesting a new technique which is based on "weight" and "matrix" to discover design patterns from of source code. In specific, the system structure is represented in columns and rows matrix to be each class in the system. The every cell value represents relation among classes. The all pattern design structure is also represented into another matrix. The design patterns discovery from source code become matching between two different matrices. If the pattern matrix matches with sub-part of the system matrix, then a candidate pattern instance is found. This method consist three different phases: "Structural", "behavioral" and "semantic" analysis. In "structural analysis", this approach explores the features of pattern structure for example classes and their relationships. The outcomes of "structural analysis" are the input of "behavioral analysis" which checks run time communication between objects. This work automated the structural, behavioral, and semantic analysis in "DP-Miner" tool. Authors are present a case study on the "Java.awt" package in "JDK 1.4" with this technique and discover design patterns.

In paper [18], pattern mining method is based on the detection of "micro structures". These "micro structures" are small modules that have limited size and scope, and that can be represented as program element such as class, method, and attribute etc. In the context of these micro structure authors are introduced design pattern clue idea which is useful in design pattern reorganization. They present "A design pattern clue catalogue". In this catalogue 46 design pattern signs are describe, subdivide into the following nine categories. 1) "*Class information"*—characterizes a single class. 2) "*Multiple class information"*—Comparison among two (or more) classes. 3) "*Variable information"*—Facts about particular variable. 4) "*Instance information"*—Particular instance of a certain class. 5) "*Method Signature information"* – Identify signature of a method. 6) "*Method Body information"*—Body of any kind of method. 7) Method Set information – Whole set of method involved in the classes. 8) "*Return information"* – Various possible return modes. 9) "*Java information"*—Clues which are strictly bound to the java language. They apply these clues on four existing detection tools and determine accuracy of result.

In paper [19], design pattern recovery approach uses "AOL" ("Abstract Object Language") for describing the structure of design pattern and existing source code. Moreover this process extracts class based metrics such that "number of public, private, protected attributes" , "number of public , private, protected operation", " number of association, aggregation , inheritance relationship" of a class of source code and apply a "brute force" method to identify particular design pattern. This approach uses open source existing software as domain which are implemented in "C++" with some conditions such that size of code is in between 3000 to 11000 LOC, number of classes are in between 29 to 187, number of relationships are in between 34 to 585. As a result this approach recovers instances of "adapter", "proxy", "composite", "bridge" and "decorator" design patterns.

In paper [20], a design pattern detection tool "Ptidej" ("Pattern Trace Identification, Enhancement and Detection in Java") is proposed. In this system the extraction of design pattern is based on "micro patterns" mining and uses "constraint satisfaction" method. Moreover, this system creates 27 different "micro patterns" which are used to describe the design patterns structure. Also this system uses its "PADL meta-model" to represent an existing software source code. Afterwards, using a software metrics library, this process identifies the design pattern structure from existing software.

In paper [21], a "micro-structure" based design pattern detection approach proposed. In this process, they are considering three issues for construct design structure. First is "Elemental Design Pattern", these structure deals with basic design features. Second is "design pattern clues", these are hints for design pattern structure. And third is "Micro Patterns", these are based on common object oriented design programming concepts such that how many attributes in a class, how many method exists in a class. Moreover this process analyze "micro structure" for design pattern detection , on the basis of six facets such that "objectives", "detail level", "definition techniques", "detection techniques", "categorization", and "interdependence among elements" . Afterwards this process verify so called design pattern instances using existing design pattern detection tool (which are developed by other researchers) such that "PINOT", "FUJABA" etc. and also provide "refinement rules" for every "GoF" design pattern.

In paper [22], a tool "PINOT" is proposed that detect design pattern from existing software source code using reclassification method. This tool reclassify design pattern into five types, 1) "Patterns that are already exists in the language". 2) Patterns that are detected by "static structural analysis". 3) Patterns that are detected by "static behavioral analysis". 4) Patterns that are domain specific.5) Patterns that are only generic designs. This tool uses AST ("Abstract Syntax Tree") for structure analysis of design pattern and CFG ("Control-Flow Graph") for execution flow of design pattern. It is built in an open source java compiler "Jakies". In this tool the pattern detection is completely hardcoded and thus it is not extendable.

In [23], a pattern detection tool "SPQR" ("System for Pattern Query and Recognition") is proposed that constructed on a theorem proving method using "rho-calculus" concept. This system is based on the EDP ("Elemental Design Pattern") extraction, which are small patterns and use of it to describe design patterns. In this process structured features of existing source code are analyzed by "rho-calculus" and represent by AST ("Abstract Syntax Tree"). A tool "*gcctree2oml"* read this "AST" file and generates "XML" file for object structure features. Moreover another tool "*oml2otter"* reads this xml file and generates a structured feature-rule input file for theorem proving. And finally, this tool using "Argonne National Laboratory's", "OTTER" to find instances of design patterns.

In paper [24], researchers are proposed tool uses a three levels design pattern detection approach. The lower level finds source code information using "reverse engineering" process. This process uses "Datrix" tool that provide existing system source code in "ASCII" based representation, here "Datrix/TA" file is intermediate format. Moreover, this approach converts "Datrix/TA" file into "XML" file. Using this "XML" file, the process extract structural information of source code such that "files name", "classifiers", "generalization relationship", "attributes", "operations", "methods", "parameter", "return type", "call action", "create action", "variable use", "friendship relationship", "class" and "function". Middle level provides a repository schema of object oriented design such as "structure", "behavior" and 'mechanisms" etc. And upper level provide end user program for pattern recovery.

In paper [25], authors provide a "reverse engineering" tool that detect design pattern in Java source code. This approach detect "structural design pattern" in two phases, in first phase candidate design patterns are identified by analyzing the class diagram information such as "name and type of class", "inheritance", "association relationship" through a "*visual language parser".* In second phase, this approach verifies the design structure through a source code "analyzer". This "analyzer" checks declaration and the invocation of the methods of the classes involved in the candidate design pattern and show whether the recognized candidate patterns are correct patterns or not. This technique implemented on structural design patterns such as "Adapter", "Bridge", "Composite", "Decorator", "Façade" and "proxy".

In paper [26], authors are proposed a metrics based approach for design pattern detection in three phases. In first phase Java source code derived in "Abstract Syntax Tree' (AST) form. In Second phase, extract necessary structural information from the AST as base for specific design pattern and create predicate for each structural condition that necessary for pattern identification. And at last stage they define a threshold value for each predicate of specific pattern. If this threshold value is true then result show that pattern is exist in application software.

In paper [27], a study on reusability using design pattern, proposed by researchers. Here, 10 metrics are proposed that are key factors of design pattern and their value show the status of reusability. These metrics are: ***"Method Reuse Factor in Pattern (MRFP)"***: This metric measure the "ratio of number of inherited method to the total number of declared as well as inherited method. Where inherited and declared M are the set of inherited and declared method respectively". Higher depth of inheritance increased value of "MRFP" and improve the reusability of pattern. 2) "***Attributed Reuse Factor of Pattern (ARFP)"***: This metric measure the "ratio of number of inherited attribute to the total number of declared as well as inherited attributes". Higher value of ARFP will increase the reusability of pattern. 3) "***Total Operation of Pattern (TOP)"***: In pattern, if number of "public operation" is large then code "complexity" is high and pattern tends to low quality. And if number of "private operation" is more than pattern tends to isolation quality. Both factor effect the pattern reusability. 4) "***Total Attributes Available to a Pattern (TAP)"***: Like operation, this metric calculate "how many variable used in a pattern", if number of "public variable" is high then "complexity" increases and quality of pattern decreases. Private variable tends to pattern isolation but "protected variable" improve the quality of pattern. 5) "***Hierarchy of Pattern Metric (HPM)"***: This metric explain order of other metric used in a class. This value helps to identify similarity among classes structure. 6) "***Depth of reusability for Pattern Tree (DRPT)"***: This metric calculate the depth of "inheritance" in complete pattern that is complete tree structure of classes used in pattern. If the value of "DPRT" high then reusability of pattern is also high.7) "***Maximum Breath of pattern tree (MBPT)"***: This metric calculate the width of class structure used in pattern. If value of "MBPT" is high then "complexity" among classes is increased and quality of pattern will be decreased. **8)** ***"Size of Pattern Hierarchy (SPH)"***: This metric value calculates the sum of "DRPT" and "MBPT". To reduce the "complexity" of software and improve the quality pattern, improve the value of "DRPT" and minimize the value of "MBPT".

In paper [28] researchers are provide a study on the reusability of design pattern. In this work authors are said that re-users are find out a group of classes from existing software that are part of specific design pattern. Moreover, modified these classes and propose in form of design pattern as a reusable piece of software. They also analyze that what is more reusable unit among "Classes", "pattern", "package". In this study they present a methodology in following steps.1) Propose some research questions. 2) Select some open source existing application software. 3) Identify approaches for comparison. 4) Minimize effect of confusing features. 5) Prepare case study.6) Implement strategy of case study.7) Analyze and give outcomes.

Amit Kumar [29] proposed a method for design pattern detection which included four steps. In first step, existing software source code converts in UML format using two tools “Rational Rose” and “StarUML”. In second step, this process uses another open source tool “JAVEX” which extract structure features from existing software source code file such that “Interface”, “Classes”, “Methods”, “Variables”, “Arrays”, “Fields”, “Relationships” etc. In third step this process analyzes run time features like “Method calls” etc. And in fourth step, using outcomes of upper three steps, In this process manually inspect the instance of specific design pattern.

In paper [30], authors are proposed a theory for design pattern. They suggested there are three types of design patterns, one is “Static Structure pattern” , i.e. patterns that consider structural relationship among participating classes of design pattern such that “Association”, Aggregation”, Generalization” etc. Second is “Dynamic Behavior Pattern”, i.e. patterns that depends on collaboration between different class objects and third is “Program Specific Patterns”, i.e. patterns that uses specific keywords in program implementation. In this paper, source code of existing software represent in AST or ASG format and search the design pattern structure using “XMI” in existing code file. Moreover, this process applying “Behavioral” and implementation specific analysis. “Builder” and “Prototype” design pattern are detected in this paper.

In [45], the design pattern detection approach , consider five type of classes such as “Non Abstract class”, “Java Abstract class”, “Java Interface”, “Abstract class” of existing software and any other type of class. This process also explores six types of relationship such as “Dependency”, “Aggregation”, “Association”, “Multiplicity”, and” Inheritance”. On the basis of these parameters they create a text file and generate “AST” for design pattern. Moreover, they also use “Java Compiler tree “API” for extracting features of existing software source code. For detection of candidate design pattern this tool extract all possible permutation of classes using brute force method. This tool detects six design patterns, “Command”, “Bridge”, “Builder”, “Visitor”, “Observer”, “Abstract Factory” and as existing domain , this process consider “JHotDraw”, “Jawa Awt”, “Apache Ant” projects.

In paper [47], authors are prosed a static analysis based on “constraints” for design pattern detection. Here they focus on structure features of existing software as well as design patterns. This process considers 4 types of constraints: 1) “class level constraints” such as “inheritance”, “association”, “aggregation” and “delegation”. 2) “Method constraints” such as invoke method. 3) “Method- class relationship constraints” such as has method, return type, has parameter etc. 4) “Attributes-class relationship constraints”. To extract the information, this process applies these “constraints” on existing software and participants of design patterns. This process also define situation where variation arise in particular design pattern. Moreover, a unique “classifier” proposed for each constraint and applies a method to match the structure features between existing software and specific design pattern. Now to find out the information, this process uses AST as intermediate presentation. Authors are also developing an information extraction tool “DPET4V”. An open source tool “Drools” is used for structure feature matching between existing software and design patterns. In this paper, the experimental results are shown on two design pattern, “Composite” and “Adapter”.

In paper [53], a “CSP” based approach suggested where a set of structural attribute proposed for existing software and design patterns. Moreover, in this process, “constraints” are defined as, how the structural attributes are collaborates in design. A mapping process identifies the presence design pattern in existing software. In [54] researcher provided a sub-graph mining approach for design pattern detection. Here source code of existing software is presented by AST and UML. Afterward this process apply a graph partitioning algorithm that divide source code model graph into small module on the basis of some structural attributes such that “class”, “interface”, “abstract class”, “template class”, and 12 types of relation between classes. Moreover this process apply another algorithm that matched design pattern graph with these small module graph and identify that any module graph is identical to design pattern graph. If a match is found than it is an instance of specific design pattern.

In paper [55], researchers are suggested a detection approach that is language independent i.e. it detect design pattern in any object oriented language. Moreover in this process, existing source code is in C# and a list of criteria defines where features of each design pattern are described. Afterwards a matching algorithm prescribes which recover the instances of design pattern.

In [56] Dirk Beyer et al. suggested a tool “Crocopat” that provides a relational programming concept for source code and uses RML (“Relation Manipulation Language”) for describing the relationship between participating classes of specific design pattern . This language based on “first order predicate logic”. Moreover, as output this tool is recover occurrences of specific design pattern in existing domain.

In [57] Marek Vokac et al. provide a tool for detection of structure of design pattern in C++ domain. Here design patterns are described in graphical notation and C++ source code file describe by a tool "UNDERSTAND". Moreover this tool uses SQL for identify the design pattern in source code and got five types of design pattern such as "Observer", "Decorator", "Factory Method", "Singleton" and "Template Method".

In [58], a "weight and matrix" based approach exist for design pattern detection. Here, design pattern and existing software code are presented in "binary matrix" form. Moreover by applying a sub-matrix algorithm, this process recover design pattern structure from source code. In this method, the structural information of source code is extract by two software, "Rational Rose" and "Star UML" and generated XMI file. Another tool "SDMatrix" is calculates the weight value for structure attributes.

Another pattern mining tool "DPRE" (Design Pattern Recovery Environment), based on visual grammar language, and is proposed in [59]. It is a two stage model where in first stage, UML of existing software source code converted into visual format of class diagram. Moreover, the structural attributes of design pattern are described in a grammar language using "XPG" (eXtended Positional Grammar). In second phase this tool identifies a sub part of visual system of source code which matched with grammar rules of specific design pattern structure. Finally outcome is shows name of classes of source code that are formed detected design pattern. In [60], a tool "HEDGEHOG" reads pattern definition which represented in "SPINE" language. This language describes structural features of design pattern. In this tool, existing source codes file representing in "AST". Moreover a pattern detection process verify that a class or group of classes of source code construct structure of particular design pattern.

III. GRAPH BASED APPROACHES FOR DESIGN PATTERN DETECTION

In paper [31], authors are proposed a design pattern recovery approach based on "sub-pattern". They introduced 15 'sub-patterns" which are different structured attributes of 23 Gof design pattern [48]. The detection process completed in three phases, first phase converts source code of existing application software and predefined "sub-patterns", both are into the class relationship directed graph. In these graphs classes are represented by "nodes" and relationships are represented by "edges". This approach also assigns weights to the edges by prime number of 2, 3, 5 and 7 to represents "association", "inheritance", "aggregation" and "dependency" respectively. Second phase of this approach identifies the number of occurrences of sub-pattern in existing software graph using "sub-graph isomorphism" method. Third phase of this process merge the required "sub-patterns" for making the specific design pattern structure (some time one sub-pattern is sufficient for a particular design pattern). Finally verify the method signature of the classes that are part of specific design pattern and compare with predefine structure of standard design patters.

In paper [32], authors are proposed a tool "DesPaD" ("Design Pattern Detector") for design pattern identification based on "sub-graph isomorphism" approach. This tool generates AST ("Abstract syntax Tree") for existing software with the help of an open source tool "ANTLR". "DesPaD" uses BNF ("Backus Normal Form") diagrams to find the relationship between classes of existing software. This tool considers four types of classes ("Class", "Interface", "Abstract Class", "template" class) and 12 type of relationship. After that this process generates design patterns structure in query item format. And last, for detection of design pattern, it applies an algorithm "Subdue's Sgiso", based on "sub-graph isomorphism".

In paper [33] the proposed methodology considers two graphs, one for existing software and other for design pattern. In specific, the method works on matrices set which expressive each significant characteristics of their static structure. For the patterns detection, employ a graph similarity algorithm which takes as input both the existing software and the pattern graph and calculates similarity scores between their vertices. The main benefit of this method is the ability to detect not only patterns in their common terms but improved variety of them. The limitation of this technique is that it only calculates similarity between two different vertices, not similarity between different graphs. To solve this problem Jing Dong [10] provide another method known as "Template Matching", which calculates the similarity between two graphs sub-graphs vertices instead. Another drawback of this method is that if the source graph is a large one then it needs a lot of computation.

Researchers Manjari Gupta and Akshara Pande are proposed some techniques in [34],[35],[36],[37],[38],[39] and [40] for structural analysis of design pattern detection. All techniques are followed "sub-graph isomorphism" concept. In these process two graphs are considered, one for existing software design graph and other for structure of design pattern. Here, these graphs are obtained from UML structure diagram of source code and UML class structure of design pattern, where nodes are represents the classes and edges are shows relationship between classes. Moreover, in [34] they applying "Decision Tree" method where both graph converts in "Adjacency Matrix" and try to find out, is there any isomorphism between sub-graph of existing software and design pattern graph. In "adjacency matrix”, this algorithm considering only those nodes in which relationship exists. In this process, first extract all possible sub-graph of existing software graph. It can be found out by generating row-column elements for all the possible “permutation” of “adjacency matrix”. Second, this process take out specific design pattern relationship matrix and represent in row-column element format and then starting from root node traverse of the “decision tree” of existing software design matrix. If any row-column element matched with design pattern matrix than an instance of design pattern is found. “Decision tree” approach is applicable for all 23 Gof design patterns. This process is not uses any real application for design pattern detection. In [35] they proposed a technique that based on matrix calculation. Here specific design pattern is identifying at various depths in the directed graph of existing software. In this process existing software transforms into a rooted graph and assigning a depth value to each node of graph. Moreover specific design pattern graph is converted into matrix for each relationship included in it. The researchers have invented a new algorithm, “DNIT”, in which the edges and nodes are labeled based on the reachability graph concept to calculate a “DNIT” table. Every relationship will have its corresponding “DNIT” table. Similarly for the design pattern its corresponding “DNIT” table is constructed. Now to search for the design pattern occurrences, the same depth entries of existing software graph and design pattern is compared. This method also can detect both incomplete and complete match of patterns. In [36], the authors are detecting design patterns in “Geographical Information Systems (GIS)”. Moreover, they draw the equivalent UML diagram for GIS application and try to find out whether a particular design pattern exists in that application or not by applying graph matching technique. In UML diagram, this process considers two relationships “aggregation” and “generalization”. Moreover these relationships are representing in “graphs” and “matrix” form. The process is complete in three steps. First step measures the graph distance between two graphs. In Second step, this process using “Normalize Cross Correlation”, to find similarity between matrixes of both graphs and in third step this process using “sub-graph isomorphism” approach to determining whether a design pattern graph is isomorphic to a sub-graph of GIS application graph. In [37], the detection process uses the “relational view” of “sub-graph isomorphism” to detect design pattern in the existing source code. A method of “n-ary” relations for “sub-graph isomorphism” is applying and find out whether a relationship graph of the design pattern exists in existing software graph. This approach detects sub graph isomorphism for each relationship between classes separately. In [38] authors are proposed an algorithm for design pattern mining using “Normalized Cross Correlation” (NCC) method. The NCC method has been commonly used to evaluate the degree of similarity or dissimilarity between two images. Moreover, this algorithm applying relationship graph and their corresponding matrix for both UML design. For calculating the value of NCC, the algorithm finds match between design pattern matrix and existing software matrix. If value of NCC for specific design pattern is matched with value of NCC of sub-graph of existing software than an occurrence of that design pattern is found in existing software. In [39], detection method is based on graph decomposition. Graph decomposition is applied on existing software graph and decomposes it into two node or three node sub-graph and find out that any sub-graph of existing software graph is isomorphic to candidate design pattern graph. In [40], detection algorithm is based on “Boolean function”. Here both graphs are converts into “Boolean Function” in “sum of product” (SOP) form. If the value of “SOP” of design pattern graph matched with any sub-graph of existing software graph than algorithm identify an instance of that specific design pattern. This algorithm cannot identify those design pattern which have a relationship within class and also consider one graph for each relationship i.e. more than one graph for specific design pattern. Manjari Gupta and Rajwant Singh Rao are also proposed some detection algorithm [41], [42] and [43] based on same concept. In [41], they are proposed a detection algorithm which is based on “inexact graph matching”. This algorithm considers situations where design pattern graph is not exact matched with sub-graph of existing software. Moreover, the “inexact graph matching” concept tolerates some missing node or relationship in graph matching and it is called “error correcting” or “error accepting sub-graph isomorphism” where error is define in term of distance between two graphs. Now, algorithm calculates error in two phases, first phase calculates error in node mapping in both graphs and second phase calculates error in edge mapping between both graphs. Finally it listed least error mapping in both graphs and identified design pattern instances. Both authors are suggested another approach in [42], the detection method is combination of “genetic algorithm” and “multilayer perceptron”. Here this method is find out whether design pattern graph matched (completely or incompletely) to any sub-graph of existing software graph by using “multilayer genetic algorithm”. “Chromosome” structure of this “genetic algorithm” is defined by node to node mapping between existing software graph and design pattern graph. The main aim of this algorithm is to find out “chromosomes” that signify “sub-graph isomorphism” between both graph. The fitness of “chromosomes” is calculates by cost of node mapping and cost of edge mapping. Moreover, the value of “fitness function” shows how much the mapping is close to the “sub graph isomorphism” between the design pattern graph and the existing software graph. After “crossover” and “mutation” operations, the terminating condition may be chosen as number of generations and find out best mapped "chromosomes" as instances of candidate design pattern. In [43], the detection algorithm calculates the value of some attributes of both graphs such that 1) "Predecessors of node" (n): set of nodes of graph from which a branch originate that ends in n. 2) "Successors of node" (n) : set of nodes of graph that are the destination of a branch starting from n. 3) "Out terminal set": the set of nodes of graph that are not in mapping but are successors of a node in mapping. 4) "In terminal set": the set of nodes of graph that are not in mapping but are predecessors of a node in a mapping. Moreover, algorithm also calculates four states for following condition.1) "Out-terminal set" of both graphs are not empty. 2) "Out-terminal set" of both graphs are empty but in-terminal set are not empty. 3) All in-terminal and out-terminal sets of both graphs are empty.4) If only one of the in-terminal set or only one of the out-terminal set is empty then state(s) cannot be part of a matching. Afterwards, the matching algorithm calculates a function under these states for all nodes of design pattern graph are matched to nodes of sub-graph of existing software graph. If such nodes are matched then algorithm checks in and out edge relationship of design pattern graph nodes are matched with in and out edge relationship of corresponding nodes of existing software graph. Majari Gupta also proposed a "greedy" algorithm [44] based on "multi-labeled directed graph", using same idea. This algorithm also finds out mapping that covers all the nodes of design pattern graph in the existing software graph. The benefit of this algorithm is that it can find out all occurrences of a design pattern and it may determine their variants also.

In paper [46] authors are proposed a design pattern detection algorithm using graphs. Here properties of candidate design pattern (which is detected) are describe in DSL ("Domain Specific Language") and convert it into graph. Also existing software source code represents in "AST" and then converts into graph. After generating the graphs for both existing software and design pattern, this process apply a graph matching algorithm based on "sub-graph isomorphism" followed by "depth first search". As a result this algorithm finds out the instance of design pattern in existing software.

IV. CONCLUSION

There has been a lot of research in the area of design pattern detection over the last years. In this paper we summarize design pattern detection method and tools.

Design pattern detection is one of the most important problems in reverse engineering. Reusable design may reduce design time, development cost and implementation time if detected in existing software and used properly in new application development. Furthermore ample possibility exit in research to improve software development paradigm to organize and harmonize the use of design pattern in more efficient manner to promote reuse and reusability in design phase, based on certain transformation rules and constraints, so that new software not only easy to design from design point of view but also developed software is easy to enhance.

REFERENCE

[1] S. Khwaja and M. Alshayeb, "A framework for evaluating software design pattern specification languages", In 12th International Conference on Computer and Information Science (ICIS), IEEE/ACIS, pp. 41-45, 2013.

[2] N.Nahar and K.Sakib, "ACDPR: A Recommendation System for the Creational Design Patterns Using Anti-patterns", In 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), IEEE, Vol. 4, pp. 4-7, 2016.

[3] A.K.Dwivedi, A.Tirkey, R. B. Ray and S.K.Rath, "Software design pattern recognition using machine learning techniques", In Region 10 Conference (TENCON), IEEE, pp. 222-227, 2016.

[4] I.Issaoui, N.Bouassida and H. Ben-Abdallah, "A new approach for interactive design pattern recommendation", Lecture Notes on Software Engineering, Vol.3.3, pp.173, 2015.

[5] S. Wenzel and U. Kelter, "Model-driven design pattern detection using difference calculation", In Workshop on Pattern Detection for Reverse Engineering, 2006.

[6] G.Rasool and P.Mader, "Flexible Design Pattern Detection Based on Feature Types", In Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering, pp. 243-252, 2011.

[7] G.Rasool and H. akhtar, "Discovering Variants of Design Patterns", In Journalof Basic and Applied Scientific Research, 3.1, pp. 139-147, 2013.

[8] A.Waheed, G.Rasool and S. Ubaid, "Discovery of Design Patterns Variants for Quality Software Development", In International Conference on Intelligent Systems Engineering (ICISE), IEEE, pp. 185-191, 2016.

[9] R. Ferenc, A. Beszedes, L. Fulop, and J. Lele, "Design pattern mining enhanced by machine learning", In Proceedings of the 21st IEEE International Conference on Software Maintenance, (ICSM'05), pp.295-304, 2005.

[10] J.Dong, Y. Sun and Y. Zhao, "Design pattern detection by template matching", In Proceedings of the 2008 ACM Symposium on Applied Computing, pp. 765-769, 2008.

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

[12] S. Uchiyama, H. Washizaki, Y. Fukazawa, and A. Kubo. "Design pattern detection using software metrics and machine learning", In First International Workshop on Model-Driven Software Migration (MDSM), pp. 38 ,2011.

[13] W. Ren and W. Zaho, "An observer design-pattern detection technique", In IEEE International Conference on Computer Science and Automation Engineering (CSAE), Vol.3, pp.544-547, 2012.

[14] D. Kirasić and D.Basch, "Ontology-based design pattern recognition", In International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, pp. 384-39, Springer, Berlin, Heidelberg, 2008.

[15] M. Thongrak and W. Vatanawood, "Detection of design pattern in class diagram using ontology", In International Conference on Computer Science and Engineering , IEEE, pp.97-102, 2014.

[16] F. Arcelli and L. Christina, “Enhancing software evolution through design pattern detection”, In Third International IEEE Workshop on Software Evolvability, pp. 7-14, 2007.
[17] J. Dong, S. Dushyant Lad and Z. Yajing, “DP-Miner: Design pattern discovery using matrix”, In 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07), pp. 371-380, 2007.
[18] F.A. Fontana, M. Zanoni, and S. Maggioni, “Using Design Pattern Clues to Improve the Precision of Design Pattern Detection Tools”, In Journal of Object Technology, Vol.10.4, pp. 1-31, 2011.
[19] G. Antoniol, R. Fiutem and L. Cristoforetti, “Design pattern recovery in object-oriented software”, 6th International Workshop on Program Comprehension, (IWPC'98), Proceedings In IEEE, pp. 153-160,1998.
[20] Y. G. Gueheneuc, “Ptidej: Promoting patterns with patterns”, In Proceedings of the 1st ECOOP workshop on Building a System using Patterns. Springer-Verlag, July 2005.
[21] Fontana, A. Francesca and M. Zanoni, “A tool for design pattern detection and software architecture reconstruction”, In Information sciences, Vol. 181.7, pp.1306-1324, 2011.
[22] N. Shi, and R.A. Olsson, “Reverse engineering of design patterns from java source code”, In 21st IEEE/ACM International Conference on Automated Software Engineering, (ASE'06), pp. 123-134, 2006.
[23] J. M. Smith, and D. Stotts, “SPQR: Flexible automated design pattern extraction from source code”, In 18th IEEE International Conference on Automated Software Engineering, Proceedings.of IEEE, pp.215-224, 2003.
[24] R.K. Keller, R. Schauer, S. Robitaille and P. Page, “Pattern-based reverse-engineering of design components”, In Proceedings of the 21st international conference on Software engineering, ACM, pp. 226-235, 1999.
[25] A De Lucia, V. Deufemia, C. Gravino and M. Risi, “Behavioral pattern identification through visual language parsing and code instrumentation”, In 13th European Conference on Software Maintenance and Reengineering, (CSMR'09), IEEE, pp.99-108, 2009.
[26] Stefan Burger, Oliver Hummel. “Towards Automated Design Smell detection”, The Ninth International Conference on Software Engineering Advances (ICSEA2014), pp. 428, October 12 - 16, 2014
[27] P. S. Sandhu, P.P. Singh and A. K. Verma. “Evaluating quality of software systems by design patterns detection”, International Conference on Advanced Computer Theory and Engineering, (ICACTE'08) , IEEE, pp. 3-7, 2008
[28] A. Ampatzoglou, A. Kritikos, G. Kakarontzas and I. Stamelos. “An empirical investigation on the reusability of design patterns and software packages”, In Journal of Systems and Software, Vol. 84.12, pp. 2265-2283, 2011
[29] A. K. Gautam and T. Gayen, “Recovery of Design Pattern from source code”, 2010.
[30] H. Lee, H. Youn and E. Lee, “A design pattern detection technique that aids reverse engineering”, In International Journal of Security and its Applications, 2.1, pp. 1-12, 2008.
[31] D,Yu, Y. Zhang and Z. Chen, “A comprehensive approach to the recovery of design pattern instances based on sub-patterns and method signatures”, In Journal of Systems and Software, 103, pp.1-16, 2015.
[32] M. Oruc, F. Akal and H. sever, “Detecting Design Patterns in Object-Oriented Design Models by Using a Graph Mining Approach”, In 4th International Conference in Software Engineering Research and Innovation (CONISOFT), IEEE, pp.115-12, 2016.
[33] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides and S. T. Halkidis, “Design pattern detection using similarity scoring”, In IEEE transactions on software engineering, Vol. 32.11, 2006.
[34] A.Pande, M. Gupta and A. K. Tripathi, “A decision tree approach for design patterns detection by subgraph isomorphism”, In International Conference on Advances in Information and Communication Technologies. Springer Berlin Heidelberg, 2010.
[35] A.Pande, M.Gupta and A.K.Tripathi, “DNIT--A new approach for design pattern detection”, In International Conference on Computer and Communication Technology (ICCCT), IEEE, pp. 545-550, 2010
[36] A. Pande, M. Gupta and A.K. Tripathi, “Design pattern mining for GIS application using graph matching techniques”, In 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), Vol. 3, pp. 477-482, 2010.
[37] M.Gupta and A. Pande, “Design Pattern Mining Using Sub-Graph Isomorphism: Relational View”, In International Journal of Software Engineering and its Application, 5.2, 2011
[38] M.Gupta, A. Pande, R.S. Rao and A.K. Tripathi, “Design pattern detection by normalized cross correlation”, In International Conference on Methods and Models in Computer Science (ICM2CS), pp. 81-84. IEEE, 2010.
[39] A.Pande, M. Gupta and A.K. Tripathi, “A new approach for detecting design patterns by graph decomposition and graph isomorphism”, In International Conference on Contemporary Computing, pp. 108-119, Springer, Berlin, Heidelberg, 2010.
[40] M.Gupta, A.Pande and A.K. Tripathi, “Design patterns detection using SOP expressions for graphs”, In ACM SIGSOFT Software Engineering Notes, 36.1, pp.1-5, 2011.
[41] M. Gupta, R. S. Rao and A. K. Tripathi, “Design pattern detection using inexact graph matching”, In International Conference on Communication and Computational Intelligence (INCOCCI), IEEE, 2010.
[42] R.S. Rao, M, Gupta, “Design Pattern Detection By Multilayer Neural Genetic Algorithm”, In International Journal of Computer Science and Network (IJCSN), Vol. 3, Issue 1, pp. 9-14, 2014.
[43] M. Gupta, R.S. Rao, A. Pande and A.K. Tripathi, “Design pattern mining using state space representation of graph matching”, In Advances in Computer Science and Information Technology, pp.318-328, 2011.
[44] M. Gupta, “Design pattern mining using greedy algorithm for multi-labelled graphs”, International Journal of Information and Communication Technology, Vol. 3.4, pp. 314-323, 2011.
[45] T. Diamantopoulos, A. Noutsos and A. Symeonidis, “DP-CORE: A Design Pattern Detection Tool for Code Reuse”.
[46] M.L.Bernardi, M. Cimitile, and G.A. Di Lucca, “A model-driven graph-matching approach for design pattern detection”, In 20th Working Conference on Reverse Engineering (WCRE), IEEE, pp.172-181, 2013.
[47] L. wen-Jin, P.Ju-long and W.Kang-Jian, “Research on detecting design pattern variants from source code based on constraints”, International Journal of Hybrid Information Technology, Vol. 8.5, pp.63-72,
[48] E. Gamma, R. Helm, R.Johnson, and J. Vlissides, Design Patterns Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
[49] M. Smolarova and P. Navrat, “Software reuse: Principles, patterns, prospects”, In Journal of Computing and Information Technology, 5.1, pp. 33-49, 1997.
[50] K.M. Hasan and M. S. Hasan, “A Parsing Scheme for Finding the Design Pattern and Reducing the Development Cost of Reusable Object Oriented Software”, In International Journal of Computer Science and Information Technology, Vol. 2.3, June 2010.

[51] G. Rasool and D. Streitfdert, "A survey on design pattern recovery techniques", In International Journal of Computer Science Issues (IJCSI), Vol. 8.2, pp.251-260, 2011.

[52] R.K. Priya, "A survey: Design pattern detection approaches with metrics", In IEEE National Conference on Emerging Trends In New & Renewable Energy Sources And Energy Management (NCET NRES EM), pp.22-26, December 2014.

[53] H. Alshira and H. Mohammad, "Integrating user knowledge into design pattern detection", (Doctoral dissertation, Department of Computer Science), 2015.

[54] U. Tekin, U. Erdemir and F. Buzluca, "Mining object-oriented design models for detecting identical design structures", In Proceedings of the 6th International Workshop on Software Clones, IEEE Press, pp. 43-49, 2012.

[55] A. Nagy and B. Kovari, "Programming language neutral design pattern detection", In 16th IEEE International Symposium on Computational Intelligence and Informatics (CINTI), pp.215-219, 2015.

[56] D. Beyer and C. Lewerentz, "CrocoPat: Efficient pattern analysis in object-oriented programs", In 11th IEEE International Workshop on Program Comprehension, pp. 294-295, 2003.

[57] M. Vokac, "An efficient tool for recovering Design Patterns from C++ Code", In Journal of Object Technology, Vol.5.1, pp. 139-157, 2006.

[58] A.K. Gautam and S.Diwakar, "Automatic Detection of Software Design Patterns from Reverse Engineering", In Issues and Challenges in Networking, Intelligence and Computing Technologies-ICNICT 2012, Special Issue of International Journal of Computer Application, November 2012.

[59] G. Costagliola, A. De Lucia, V. Deufemia, C. Gravino and M. Risi, "Design pattern recovery by visual language parsing", In Ninth European Conference on Software Maintenance and Reengineering, CSMR, IEEE, pp.102-111, 2005.

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