International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 11, November-2015

Design Pattern Detection and Visualization using Reverse Engineering

Jerin Thankappan
M.E. Student
Computer Department,
St. Francis Institute of Technology
Mumbai, India.

Vandana Patil
Assistant Professor
Information Technology Department,
St. Francis Institute of Technology
Mumbai, India.

***Abstract*—Design Patterns are the most valuable and approachable solution for any software design issues. Pattern identification delivers important information to the designers. The proposed work here gives a detailed analysis of this solution and also helps in fulfilling the detection of design patterns in java. Since design is not tangible, it is hard to detect. The proposed application is functioned for all kinds of java based software where any person interested in understanding the code can use. The focus here is to detect the java design patterns and display them with the help of a proper GUI for easier understanding. Functionally, this tool gives an overall idea of the design pattern and its detection.**

Keywords—Reverse Engineering, Web design patterns, java based tool, detection, tree structure, class diagram.

I. INTRODUCTION

The secondary developers, when they try to reinvent the wheel for any software face problems related to the design of the product. These problems are related to the design patterns used in the software. The major task in software comprehension is to understand the design and architecture [3]. For a program comprehension it is important to discover the uncovered design patterns from source code. The intended design is totally different from what you expect, so there is a need to understand these differences and provide a solution to the concept being used here. Dirk Heuzroth et.al have provided with static and dynamic information in their tool. Static information includes the structural connections among classes via call, delegation or inheritance relations. Dynamic relations are the ones where the specific sequences of actions and interactions of the objects of these classes are shown. The lack of a common structure of the architecture causes problems in understanding the code [4]. These patterns are usually in the textual format, which is semi-formal way to be precise. Since graphical images win over text, the need to produce it graphically was embarked. The usability and quality of the interface should be evaluated for its conformity. The web design patterns deal with interaction problems of web application users. The focus of this paper is to provide with an optimal solution to detect web design patterns in java source code in a graphical method much efficient and clear to understand. Hence the need to give a good graphical presentation for the detected patterns was discovered. This paper uses similarity scoring for graphs to compare and detect the pattern. We use reverse engineering process to detect these design patterns. It is important to document the code for many purposes such as for analyzing the product, for provides any updates or correction, for commercial or military espionage, for academic purposes, for gaining technical intelligence or for the curiosity of the learner.

II. LITERATURE REVIEW

Design patters which are the elements of reusable object oriented software was originally written by the Gang of four (GoF) viz Erich Gamma et.al. They have categorized the design patterns into 3 viz. Creational, Structural and Behavioral. Creational addresses problems of creating an object in a flexible way, structural addresses problems of using object oriented constructs like inheritance to organize classes and objects and behavioral address problems of assigning responsibilities to classes, they suggest both static relationships and patterns of communications. The diagram shown below clearly shows those patterns.

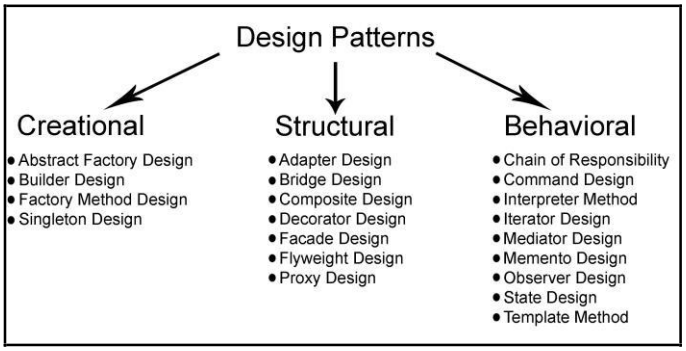

Figure 1. Classification of Design Patterns (GoF) f[2]

In [3] Dirk Heuzroth et al., propose to analyse the system and also consider the structure the behaviour of the given system. In this paper they take an example of the Observer Pattern and say that it is possible to distinguish an object instance from a design instance, if the latter contains class program elements. Example of Observer pattern is used to distinguish the object instance from its design instance. They distinguish the static and dynamic restrictions or rules. The below figure shows the approach

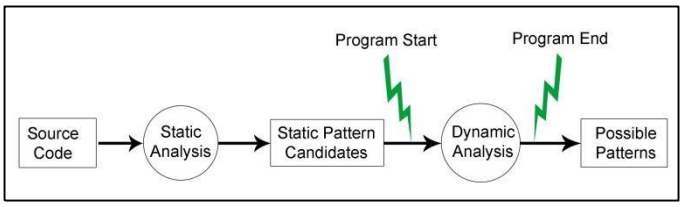

Figure 2. Static and Dynamic Analysis of Patterns [2]

Dirk Heuzroth et al., [3] also mentions about other tools which are available that describe a static analysis to discover design patterns (Bridge, Factory Method and Template Method) from C++ systems. The Pat system mentioned in the paper detects structural design patterns by extracting design information from C++ header files and they store them as Prolog facts. Here the Prolog queries are used to search and the patterns are expressed as patterns. In a paper Brown K. et al. uses dynamic information to analyse the flow of messages. This approach is restricted to detecting design patterns in Smalltalk. Carriere et al. also works on code instrumentation to extract dynamic information to examine and transform architectures. The approach presented by Dirk H. only recognizes communication primitives and not complex protocols.

In [4] Nazanin A. et al., inform that all patterns are documented in natural language and also reproduced via diagrams. Architectural patterns use an simple outline for visualization purpose, SE patterns make use of UML diagrams to illustrate design patterns in this field. HCI design patterns face problems in visualization and uncertainty. The usage of UI Model fragments is referenced in Lucca, Fasolino and Tramontana [4]. They provides an effective definition for HCI design patterns. Since this method is not descriptive enough, it cannot be used for visualization purpose and it only helps for detection. Till now, a great amount of research has been done into detecting design patterns but majority of it has been done in the field of HCI. In [6], Shi and Olsson addressed only Structural and Behavioural aspects. They divided design patterns detection methods into static and dynamic analysis approaches. Among the approaches developed for detection of patterns, Dong, Sun and Zhao referred in [4] and Shi and Olsson [6] used Matrix Matching method and reorganization of GoF patterns to consider only static analysis for detection of design patterns. Also, Heuzeroth, Holl, Hogstrom and Lowe [3] examined automatic detection of design pattern through the usage of both static and dynamic analyses. It can be noticed that the focus of reviewed approaches is either on detection or visualization of design patterns. We notice that the focus of the papers seen above is either on detection or on visualization.

According to Kniesel, Günter, and Alexander Binun [9] their approach brings data fusion to software engineering. They are the first one to study an approach to detect the design pattern based on data fusion. Poshyvanyk et al. as referred in [9] suggest using data fusion to find features in big programs. They suggest to combine the outputs of the static and dynamic analysis-based concept detection tools. Kniesel, Günter, and Alexander Binun went one step further by combining the output of tools that already combine different techniques. The decomposition of patterns into smaller subparts has already been suggested in SPQR and implemented in SPQR, Fujaba, EDPDetector4Java(tools mentioned in [9]) and to a limited extent (one EDP) in SSA. They take this decomposition one step further by showing that also much bigger entities can beneficially be used as witnesses of sought patterns. Our witness relation was based on our sub-pattern relation. However, other relation between patterns has not been discussed and has been kept for future work. The difference between the above approaches and ours is that we emphasized that roles need to be regarded at any granularity, not just at the level of classes. The approach in [9] is partly based on performing two analysis stages. There are undoubtedly many parallels of our work to any other work on Design Pattern Detection. In this paper we have thoroughly discussed some of the currently available tools and compare our results with theirs. For a broader discussion of other approaches we refer to the recent state of the art overview by Kniesel et al[9] and Dirk Heuzroth[3]. However, most tools were compared only based on literature, without running them. Only [9], did practical experiments on the tools, rest of the tool evaluation was purely on literature. In this paper too, we are publishing the results based on the experiments that were conducted with other tools and our tool called the "CLASSIFIER". Our experimentation enabled us to provide detailed feedback to the various tool authors, regarding the robustness, performance, scalability, precision and recall of the approaches.

III. PROPOSED SYSTEM

A design pattern is a concept of source code design and it can be found out in many ways. As we saw above there are many tools which analyse design pattern and detect them using different tools. However, the presentation techniques are different and some of them do not provide accurate results. This detection is very important as it becomes difficult for the secondary developers to make any changes. Hence, reverse engineering of these applications becomes a necessity and has to be taken care of. The knowledge of this design pattern improves the program understanding and software maintenance. Therefore, an automatic and consistent design pattern discovery is required. This System detects the pattern by using a tree structure, this tree structure will be the basis of comparison and along with the detected pattern we will also have the tree structure shown where all the defined methods and classes will be mentioned shown on the tree as a diagrammatic representation so that the secondary developers may not waste time searching and understanding the documentation of each software.

The system here we call as Classifier, detect the patterns using similarity scoring and create a tree structure for the developers to understand the code in a better manner. The components include a parser and a comparer. The parser parses the java code given to it and the comparer compares the tree structure to the existing structure and produces the result. First, a java code has to be given as the input to the system, and then the system generates a tree structure of the code. This structure will contain all the classes as given in the code along with the class definition and instances if any. Here we use the similarity scoring algorithm for graphs [10] and detect the patterns. There does not exist a design pattern always, hence for such codes, the pattern may show none, but the tree diagram will be produced displaying all the classes and methods.

IV. EXPERIMENTAL RESULT

The proposed system uses different method to detect the web patterns. The experiment was conducted on a desktop computer running on Windows 8 Home Basic Operating System with 2GB RAM and 500GB Hard disk. The program was run on Spring Tool Suite IDE for the integration of Web with Java. Spring tool has full Eclipse JEE distribution inside and comes with Maven, Spring Roo and tc Server developer edition pre-packaged and pre-configured (you can start using tc Server or Spring Roo right away without the need to download or configure them manually).

A. Data

The data or the java source files for testing purposes was used. These examples are based on specific patterns. We provide with 2 such examples of Observer and Composite Patterns. Let us take this code snippet for example [14].

```
interface AlarmListener { public void alarm(); }

class SensorSystem {
  private java.util.Vector listeners = new java.util.Vector();

  public void register( AlarmListener al ) {
listeners.addElement( al ); }
  public void unregister( AlarmListener al ) {
listeners.removeElement( al ); }
  public void soundTheAlarm() {
    for (java.util.Enumeration e=listeners.elements();
e.hasMoreElements(); )
      ((AlarmListener)e.nextElement()).alarm();
        }
}

class Lighting implements AlarmListener {
  public void alarm() { System.out.println( "lights up" ); }
}

class Gates implements AlarmListener {
  public void alarm() { System.out.println( "gates close" ); }
}

class CheckList {
  public void byTheNumbers() {  // Template Method design
pattern
    localize();
    isolate();
    identify(); }
  protected void localize() {
    System.out.println( "   establish a perimeter" ); }
  protected void isolate()  {
    System.out.println( "   isolate the grid" ); }
  protected void identify() {
    System.out.println( "   identify the source" ); }
}
            // class inheri.  // type inheritance
class Surveillance extends CheckList implements
AlarmListener {
  public void alarm() {
    System.out.println( "Surveillance - by the numbers:" );
    byTheNumbers(); }
  protected void isolate() { System.out.println( "   train the
cameras" ); }
}

public class ClassVersusInterface {
  public static void main( String[] args ) {
    SensorSystem ss = new SensorSystem();
    ss.register( new Gates()       );
    ss.register( new Lighting()    );
    ss.register( new Surveillance() );
    ss.soundTheAlarm();
        }
}
```

B. Result

We find that the AlarmListener interface is been called by class Lightning, Gates and Surveillance, and Surveillance also extends CheckList, which follows the observer pattern.

This code when fed as input to CLASSIFIER and WOP (Web of Patterns), both the systems detected the pattern correctly, but the representation was different. WOP showed rigid formation and limited information with respect to class and methods whereas CLASSFIER showed a GUI which can be shaped according to user's comfort and well presented with proper address to methods in each class. The Classifier diagrams are more accurate and understandable than other systems. The diagram below shows the result for classifier.

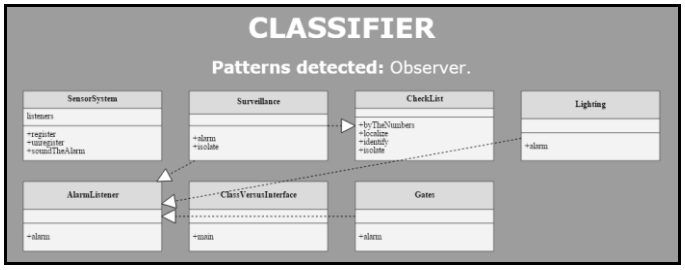

Fig 3. Result from the given code

The class boxes can be moved as per the user's understanding and flexibility which is shown in Fig. 4.

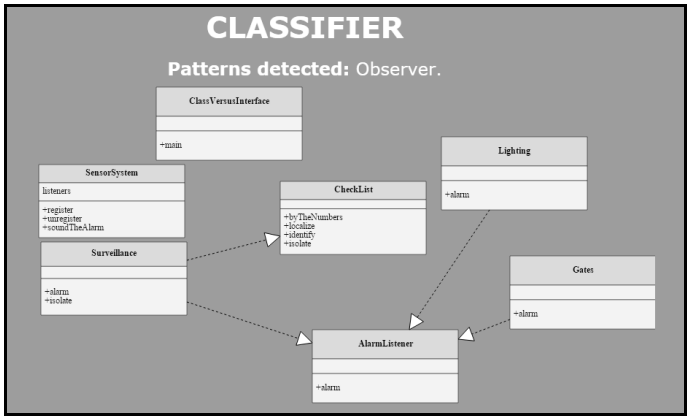

Fig 4. Reshaped Classes

V. CONCLUSION

Classifier is a system which detects the design patterns in any java source code provided if it exists. The system first creates a tree structure of the code entered and then compares it with the existing patterns in the system. When you compare the work with any other existing tool the accuracy level is more and the pattern detected are represented using a flexible GUI which makes it easier for the developers to adjust and makes changes as how one wants to view it. The detection is done using similarity scoring for graphs which is open-source and easily available. For future work one can extend this to design patterns for web, i.e. html pages and can create similar structure for the css and javascript involved.

REFERENCES

[1] Jerin Thankappan, and Vandana Patil. "Detection of Web Design Patterns Using Reverse Engineering." Advances in Computing and Communication Engineering (ICACCE), 2015 Second International Conference on. IEEE, 2015.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns–Elements of Reusable Object-Oriented Software", Addison-Wesley, Boston, Massachusetts, USA, 1995

[3] Dirk Heuzroth, Thomas Holl, Gustav Hogstrom, Welf Lowe, "Automatic Design Pattern Detection" IEEE Conference Publications, 2003

[4] Nazanin Aminzadeh, Siti Salwah Salim, "Detecting and Visualizing Web Design Patterns",Computer and Automation Engineering (ICCAE), the 2nd International Conference,2010

[5] Osama Abu Abbas, "Recovering Interaction Design Patterns in Web Applications", Software Maintenance and Reengineering, Ninth European Conference, 2005

[6] Nija Shi and Ronald A. Olsson, "Reverse Engineering of Design Patterns from Java Source Code" Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on. IEEE, 2006

[7] Federico Bergenti and Agostino Poggi, "IDEA: A Design Assistant Based on Automatic Design Pattern Detection" Proceedings of the 12th international conference on Software Engineering and Knowledge Engineering, 2000

[8] D.H. Qiu, H. Li, J.L. Sun, "Measuring Software Similarity based on Structure and Property of Class Diagram" Sixth International Conference on Advanced Computational Intelligence, 2013

[9] Kniesel, Günter, and Alexander Binun. "Standing on the shoulders of giants-a data fusion approach to design pattern detection." Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on. IEEE, 2009.

[10] Chatzigeorgiou, Alexander, Nikolaos Tsantalis, and George Stephanides. "Application of graph theory to OO software engineering." Proceedings of the 2006 international workshop on Workshop on interdisciplinary software engineering research. ACM, 2006.

[11] Pinot–Pattern Inference and Recovery Tool. http://web.cs.ucdavis.edu/~shini/research/pinot/

[12] The Web of Patterns Project. http://www-ist.massey.ac.nz/wop/

[13] Fujaba–Tool Suite. http://www.fujaba.de/

[14] Observer Design Pattern in Java. https://sourcemaking.com/design_patterns/observer/java/2