A FRAMEWORK FOR RECOGNATION AND MODIFICATION OF DESIGN PATTERNS

Surendra Shukla*

Rashmit Kaur Khanuja*

Reena Karandikar**

ABSTRACT

Today, The requirements of market and software technology are changing dynamically, software systems are growing rapidly and changing. The design pattern play an important role for documentation of software systems architecture. Thus, there is a need in computer science and industry for recognizing these patterns. In this paper we present a framework that recognize design pattern from the source code and modify it on the basis of customer need. The framework required some precise information like inheritance, composition, aggregation and association for modification in design pattern.

***Keywords**: Design pattern, DPML, C++, UML.*

*Assistant Professor, Department of Computer Science & Engineering, Chameli Devi School of Engineering, Indore, M.P.

**Lecturer, Department Of Computer Science & Engineering, KC Bansal Technical Academy, Indore MP, India.

I. INTRODUCTION

Software Systems are rapidly growing and changing, so the source code written today becomes legacy code in very short period of time, this is because of changing requirement of the market, and changing of technology, because of tight deadlines developers some time not releases the software in proper format (design description and source code).In these cases the valid document remains only the source code itself. Re engineering solves these problems mentioned above. The first part of re engineering is the reverse engineering, which deals with identifying the system components and converting these to another form. Object oriented technology is mainly used now a days for developing the software the systems developed on these technology now have became the legacy projects. And all these projects are needed to re engineered. Complex telecoms projects are developed in language c++, and this is the most complex language, and not supports the re engineering tool's this language gives us the lot opportunity to develop the reverse engineering tools. For developing the tool we needed to develop the miners, who will basically identify some facts from the source code. That we mine in the source code is the design patterns. Source code now a days are build by the design patterns, so for the reverse engineering we need to identify the design patterns. So we need to know about design patterns.

Design pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice. Design pattern can be the measure of the quality of object oriented software system so one can characterize the software system by no. of design pattern used. Without fully understanding the design pattern if some one will use this, it may create a huge class structure which will decrease the software quality. There are three kinds of design patterns, recreational, structural, behavioral.

Creational: These Patterns are concerned with the creation of objects. They can decide the type of object and its multiplicity aim of this type of design pattern is that client must not have to be aware,how object is going to be created. This responsibility must have to be taken either the class or by the some methods. We can identify these design patterns because by analyzing the source code it is not impossible.

Structural: These patterns deal with the composition of classes or objects. They define class hierarchies and different relations. In the category of these design patterns most features are described with the declarations of operations and attributes,so they are easier to recognize.

Behavioral: These patterns describes how objects interacts to each other, how they behaves at the time of interaction. They also describe how classes interact and distribute responsibility. As we know behavior is described with the help of the operations, and operations are defined in the bodies of the operations, knowledge of declaration is insufficient. Because of this identification of these patterns is very difficult.

To know about the unknown system we will have to extract various facts about it. We will say it fact about the source code eg. Size of source code is fact, whether a class have a base class we can collect all the facts by hand but if the size of the source code is small. Real world projects have million lines of code so it can be processed only by the software tool.

II. RELATED WORK

1. Kramer et al. Used the pat system it works on the out put of oo CASE tool which converts it on prolog facts, it gives the structural analysis of the code based on c++ header files
2. The second work describes a method based on multi stage reduction strategy using software metrics and structural properties to extract structural design patterns from oo design model or sources code. Code and design are mapped in the intermediate representation called abstract object language.
3. The third work is done with the integration of two existing tools called Columbus and maisa.columbus is a extraction tool and maisa is clause based mining tool. c++ code is analyzed by the Columbus and a plug in was written for it understandable by the maisa.
4. DP++ was also detects the most structural patterns
5. PTIDEJ (Pattern trace identification,detection and enhancement for Java) is based on analyzing Java AWT and net libraries,and it founds the occurrences of the facade and composite design pattern.

III. PROBLEM DOMAIN

For Doing the reverse engineering we will have to be familiar with the existing system in detail.

To know about the unknown system we will have to extract various things about it. We refer this information as facts about the source code eg. Size of source code is fact, whether a class have a base class is also the fact. We can collect all the facts by hand but if the size of the source code is small. Real world projects have million lines of code so it can be processed only by the software tool.

IV. SOLUTION DOMAIN

Design Pattern mining is a process where structure of the design pattern is searched in the source code. Structure must have to be flexible and short, so that it can be matched with the structure found in the source code.

For the easy description of the structure of design pattern we are using XML based language. With this we can easily describe the design pattern. Language is easy to understand and descriptions are easy to modify. The search for the pattern is performed by matching the pattern classes(pattern description) to source classes(source code).when we will extract some pattern from source code for that source code we can have number no candidate pattern classes(pattern description). Search is divided into two stages, first stage: filtering the candidates (source classes) for pattern classes second stage: source classes are bound to the pattern classes, and all the constraints are checked to see that if any pattern is found with this.

DPML: Design pattern markup language

our algorithm uses the XML based language for the design pattern description

Proxy Design Pattern in DPML

```
01<?xml version='1.0'?>
02<!DOCTYPE DesignPattern SYSTEM 'dpml-1.6.dtd'>
03
04<DesignPattern name='Proxy'>
05
06 <Class id='id10' name='Subject' isAbstract='true'>
 <Operation id='id11' name='Request' kind='normal'
07  isVirtual='true' isPureVirtual='true'>
08    <hasTypeRep ref='id50'/>
09  </Operation>
10
11 </Class>
12
13 <Class id='id20' name='Proxy'>
    <Base ref='id10'/>14
14    <Aggregation ref='id30'/>
15<Operation id='id21' name='Request' kind='normal'
16            isVirtual='true' isPureVirtual='false'>
```

```
17   <defines ref=’id11’/>
18      <calls ref=’id31’/>
19  <hasTypeRep ref=’id50’/>
20   </Operation>
21   <Attribute id=’id22’ name=’realSubject’>
22   <hasTypeRep ref=’id52’/>
23  </Attribute>
24
25 </Class>
26
27 <Class id=’id30’ name=’RealSubject’>
    <Base ref=’id10’/>
28   <Operation id=’id31’ name=’Request’ kind=’normal’
29              isVirtual=’true’ isPureVirtual=’false’>
30     <defines ref=’id11’/>
31       <hasTypeRep ref=’id50’/>
32  </Operation>
33
34 </Class>
35
36<TypeRep id='id 50'>
37<TypeFormerFunc>
38<hasReturnType ref='id51'/>
39</TypeFormerFunc>
40</TypeRep>
 41</Design Pattern>
```

V. APPLICATION DOMAIN

1. Identification of design patterns in legacy code.
2. Identification of design patterns in minimum time.
3. Providing options for more than one object oriented programming code.
4. Developing platform independent and open source tool.
5. Description of the design pattern.

This section must provide scope of the work. Where it is applicable. It should also depict the variants if any and impact of the work on real life / end user.

VI. EXPECTED OUTCOME

1. A tool for Mining the design pattern from source code
2. Supporting at least 2 Object oriented programming languages whose source code can be Mined.
3. For detecting the design pattern will take less time.
4. Support for suggestions that if some design patterns are not identified then where to make changes in design pattern description

VII. CONCLUSION

Design pattern detection is the requirement of now a days in software industry. There are number of algorithms, concepts are given by various researchers for design pattern detection. They all are different in, how many design patterns they are detecting, and what amount of time they are taking in future we can make a tool who can compare various algorithms for design pattern detection.

VIII. REFERENCES

1. J. Bansiya. DP++ is a tool for C++ programs. In Dr. Dobb's Journal, June 1998.
2. K. Brown. Design reverse-engineering and automated design pattern detection in Smalltalk. In Master's thesis. Department of Computer Engineering, North Carolina State University, 1996.
3. R. F. G. Antoniol and L. Cristoforetti. Using Metrics to Identify Design Patterns in Object-Oriented Software. In Proceedings of the Fifth International Symposium on Software Metrics (METRICS98), pages 23–34, Nov. 1998.
4. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Pub Co, 1995.
5. Object Management Group Inc. OMG Unified Modeling Language Specification, version 1.3 edition, 1999.