Automatic Detection of Software Design Patterns from Reverse Engineering

Amit Kumar Gautam
IMS Engg. College, Ghaziabad, India

Saurabh Diwaker
KIET, Ghaziabad, India

ABSTRACT

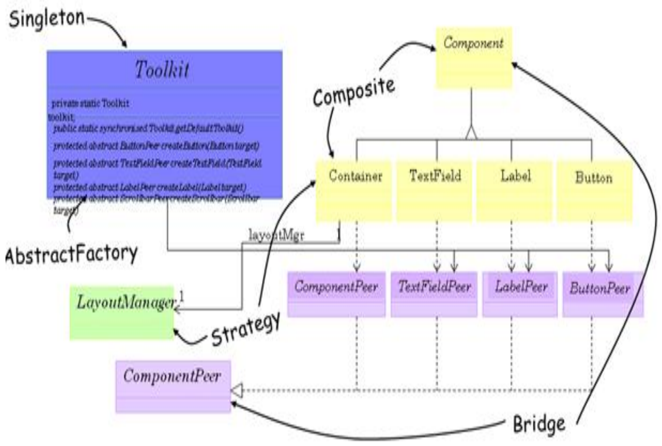
It is proposed to present a novel approach to recover design patterns which can achieve better performance and greater accuracy by representing the characteristics, basically structural, behavioural etc. of design pattern by using weight and matrix concept so that to reduce the anomalies like false positives rate and false negative rate. Also follow the pattern taxonomy for reverse engineering and applying sparse matrix algorithms for efficient storage and computation. Apply the sub matrix algorithm to design pattern binary matrix and binary matrix generated from source code. Comparison with other standard pattern detection tools for effectiveness and performance.

Keywords: XMI file, matrix matching, SD Metrics

1. INTRODUCTION

Design patterns are widely used in every domain of software industry, to identify the problem that is similar for many different categories of software and trying to build a reusable solution which can be reused in different types of environment and context. It is a common general reusable solution to an iterative problem in design [3][4][5]. They are so distinguishing in nature in many ways that it is intend for a problem and is independent of particular domain and technology. Design pattern provide developers with base for development of object oriented frameworks and toolkits which is extensively used in component development as well as component based software development. Design patterns tell us the way of structuring classes and objects to solve certain feasible problems and it's our responsibility to follow and acclimate those designs to suitable our particular application. It can also handle both functional as well as non functional requirement i.e. quality attributes. Patterns neither provide exact solution nor it solves all the design problems but it captures essentials parts of design in compact form with one or more solution. A design pattern is visualized and described by the four mandatory elements

- Pattern Name, brief description of problem, its solution, and consequences
- Description of problem (usage scenarios) describes when to apply the pattern.
- Description of solution (involved objects and their interaction behavior, interfaces generalization, aggregation, realization).
- Consequences are the outcome and trade-offs of applying the pattern


Singleton
Toolkit
private static Toolkit toolkit;
public static synchronised Toolkit getDefaultToolkit()
protected abstract ButtonPeer createButton(Button target)
protected abstract TextFieldPeer createTextField(TextField target)
protected abstract LabelPeer createLabel(Label target)
protected abstract ScrollbarPeer createScrollbar(Scrollbar target)
AbstractFactory
Composite
Component
Container
TextField
Label
Button
layoutMgr
ComponentPeer
TextFieldPeer
LabelPeer
ButtonPeer
LayoutManager
1
Strategy
ComponentPeer
Bridge

The above figure represents the various design patterns in a particular UML diagram.

Design pattern does not solve all the design aspect and issues but solve the most critical aspects of design which is essential for further modifying the software. It describe core of the solution of that problem. It is basically description of classes, interfaces and communicating objects customized in such a manner to solve a general design issue in a particular context. Design principles provide certain rules then an object oriented software should consider during the software design phase in order to make

.A design pattern are

A. Smart

A novice would not think of it quickly

B. Generic

A pattern is meant for a problem and does not depend on a domain, platform or technology

C. Well proven

Identified from real systems that have been applied several times

D. Simple

Usually quite small with just a few classes

E. Reusable

Design patterns are well structured and documented and so can be used in different lexicon.

- Reusable solutions to common problems
- Names of abstractions above the class level

Based on a design of experiments in [3] it is proven that documented patterns lead to an easier and better understanding of software systems. Large computer-based systems and legacy systems are normally difficult to understand, extend and maintain due to lack of software architecture and design documentation. After the integration, packaging and deployment of the software based systems, the original software architecture and design related information, and experience of expert designers is generally lost [1]. Source code becomes the only source to understand and further enhance the systems. However, source code is very large in size and hard to comprehend. It is very critical, time-consuming and error prone process to observe source code manually. Understanding the existing legacy systems, extracting and factoring out the relevant design information is very essential since it may help on modifying them and to facilitate the software development life cycle like spiral model, incremental model, prototype model where there is need of automation of existing manual system. By extracting the design pattern we can also able to reconstruct the original software architecture of the different modules and components.

Software systems generally should be amenable to changes due to continuous changes of user requirements, domains, platforms, technologies and environments. Requirements are dynamic in nature so change is a constant theme of computer-based system design and development. To analyze and understand the source code of existing computer-based system, we need to discover the original architectural, design decisions and tradeoffs. Maintenance is one of the necessary and critical aspect of the software evolution since it involves lot of operational cost and therefore requires lot of proper documentation like software requirement specification document. In maintenance phase, we require lot of documentation which is generally lost after deployment otherwise it leads to huge complexity of reverse engineering. Generally, design-pattern is a reusable solution so by identifying it we can construct high level design like SRS document as well as low level design. If design-patterns could be identified and reused in reverse engineering, the process of doing reverse engineering would be very effective and helpful to those people who are involved in system designing and maintaining the software. So there were many attempts to detect design-patterns during reverse engineering. Design pattern are higher level of abstraction than libraries. Frameworks and libraries are not design patterns and there does not exist any libraries of design patterns.

Design patterns are also used in the process of designing and maintaining component-based systems for which the reusable modules must be found out. By discovering design patterns from reusable software, it is easier to recognize and verify those reusable parts in legacy software based system or figure out in the form of pre-developed components, or build them as reusable product. Extracting the architectural and design information from legacy software which consists of large number of classes i.e. large search space is very typical. So software metrics and optimization techniques help us to reduce the search space to great extent.

2. LITERATURE SURVEY

An Since 1990s researchers have been working on “Recognizing Programmers Design” [1-4]. After popularity of design patterns, since 1996, object oriented design community began to collect design patterns used in the software construction. Since, a pattern provides knowledge about the role of each class within the pattern, the reasons of certain relationships among pattern constituents and/or the remaining parts of a system, localizing instances of the design patterns in existing software, can improve maintainability of software with other benefits like code comprehension, analysis of effects of using design patterns in software development. Some approaches [5-25] have been proposed to detect design patterns from source code or a design model, such as the UML diagrams. Even then to date little research has focused on the development of techniques for discovering design patterns. Our work is a step in this direction.

There are many existing methods for discovering design patterns from design and source code. Rudolf et. al. [5] presented a pattern matching-based system using the Columbus framework with which they were able to find pattern instances from the source code by considering the patterns’ structural descriptions, but with this method they could not identify false hits and distinguish similar design patterns such as State and Strategy. Then they used machine learning algorithms, such as decision tree and neural network, to enhance pattern mining by filtering out as many false hits as possible [5]. To do so they distinguish true and false pattern instances with the help of a learning database created by manually tagging a large C++ system. Ozalp Babaoglu et al. [6] proposed design patterns as a conceptual framework for transferring knowledge from biology to distributed computing. The motivation of their work is that large-scale and dynamic distributed systems have strong similarities to some of the biological environments. This makes it possible to abstract away design patterns from biological systems and to apply them in distributed systems. They did not extract design patterns from software engineering practice, as it is normally done. Instead, they extracted design patterns from biology and argued that these can be applied fruitfully in distributed systems. In [7] Pree’s meta patterns are used to represent the common properties of design patterns as a part of the detection conditions.

A template matching method [8] from computer vision has also been applied by calculating the normalized cross correlation between pattern matrix and system matrix. Graph theory [9-12] has also been applied in detection of design patterns by ascertaining similarity between the classes (vertices) in different systems (graphs) using the similarity score and iterative algorithm. Kleinberg [11] proposed link analysis method to find the main hub and source nodes for web pages. Blondel [10] generalized this idea to an iterative algorithm for computing the similarity score for any two vertices's. This similarity score algorithm for design pattern detection has been applied in [9] by encoding the source code and design patterns into different feature matrices. Kramer and Prechelt [13] have proposed an approach and developed a system, called Pat, to localize instances of structural design patterns, extracting design information from a CASE tool repository and using Prolog facts to represent it and rules to express patterns. Antoniol et al [14] proposed a conservative approach, based on a multi-stage reduction strategy, using software metrics and structural properties to extract structural design patterns from OO design or code. Code and design are mapped into an intermediate representation, called Abstract Object Language.

Antoniol et al [15] presented a approach in which a design pattern is represented as a tuple of classes and relations among classes. OO software metrics are used to determine pattern constituents candidate sets to avoid combinatorial explosion in checking all possible class combinations. Pattern structure is then exploited to further reduce the search space. Shull, Melo and Basili [16] have developed an inductive method to help discovering custom and domain-specific design patterns in existing OO software systems. The method however is performed manually, although it could be greatly assisted by tools. Different approaches, exploiting software metrics, were used in previous works to automatically detect design concepts and function clones [17] in large software systems.

Bergenti et al. [18] presented a system called IDEA (Interactive DEsign Assistant). IDEA is an interactive design assistant for software architects meant for automating the task of finding and improving the realizations of design patterns. IDEA is capable of automatically (i) finding the patterns employed in a UML diagram and (ii) producing critiques about these patterns. The core of IDEA is the module that automatically detects the pattern realizations found in the model that the architect is producing. When this module finds a pattern realization, a set of design rules are verified to test if the design could be improved. Any violation to these rules fires a critique that is proposed to the engineer as a possible design improvement. Currently, a prototypal [18] implementation of IDEA is integrated with two popular CASE tools.

Stencel et al. [19] presented a method that is able to detect many nonstandard implementation variants of design patterns. They presented its proof-of-concept implementation and also compared its efficiency to other state-of-the-art detection tools. The presented method is customizable. An analyst can introduce a new pattern retrieval query or modify an existing one and then repeat the detection using the results of earlier source code analysis stored in a relational database. Dong et. al. [20] presented a novel approach to discovering design patterns by defining the structural characteristics of each design pattern in terms of weight and matrix. Their discovery process includes several analysis phases.

Their approach is based on the XMI standard so that it is compatible with other techniques following such standard. They also develop a toolkit to support their approach. Francesca et. al. [21] described an approach to design pattern detection using supervised classification and data mining techniques based on sub-components, and summarized the results they obtained on behavioural Design Patterns. Their Experiments with neural networks showed some encouraging results, but their instability led them to decision of employment of different techniques. Jing et. al. [22] presented some experiments on design pattern discovery from open-source systems using the tool they developed for design patterns detection: DP-Miner. In particular, their experiments discover the Adapter, Bridge, Strategy, and Composite patterns from the Java.AWT, JUnit, JEdit, and JHotDraw systems and experimental results show that design patterns have been widely applied in these systems and can also be recovered. In addition, they compared their experimental results with those of others and found several discrepancies. They analysed this issue and discussed possible reasons for the discrepancies. More importantly, they argue for benchmarks for design pattern discovery.

Damir et. al. [23] presented ontology-based architecture for pattern recognition in the context of static source code analysis. The proposed system has three subsystems: parser, OWL ontologies and analyser. The parser subsystem translates the input code to AST that is constructed as an XML tree. The OWL ontologies define code patterns and general programming concepts. The analyser subsystem constructs instances of the input code as ontology individuals and asks the reasoned to classify them. The recognition system is envisioned as a framework that can be used as a stand-alone utility, or as a subsystem for various larger systems, such as a compiler front end or IDE plug-ins. There are many other techniques that have been proposed earlier. The main problems encountered in using the above mentioned techniques are related to scalability, to many false positive results, and to the impossibility to find several design patterns; hence we decided to explore the problem and trying to overcome some of the mentioned or encountered difficulties.

3. PROPOSED APPROACH

The existing approaches suffer from various anomalies and none of the existing software based on reverse taxonomy is able to detect the entire static structural design pattern.

A. Proposed Architecture of Design Pattern Detection:

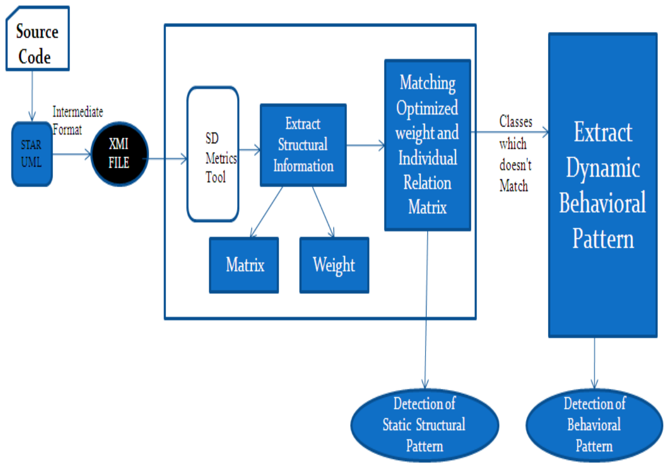

Source Code
STAR UML
Intermediate Format
XMI FILE
SD Metrics Tool
Extract Structural Information
Matrix
Weight
Matching Optimized weight and Individual Relation Matrix
Classes which doesn't Match
Extract Dynamic Behavioral Pattern
Detection of Static Structural Pattern
Detection of Behavioral Pattern

Figure 3.1: Architecture of design pattern detection

According to my proposed architecture the source code is analyzed and apply reverse engineering of java source code with the help of Star UML/Rational Rose tool to find out the class diagram from source code and using export XMI feature it can be exported into XMI file which contain the structural information which is required by the structural analysis phase for detecting design pattern.

One important tool "SDMetrics", the quality measurement tool for UML™ designs through which we can easily calculate all the structural information of different classes and packages stored in XML file and the stored in tab separated text file.

B. Design of the system

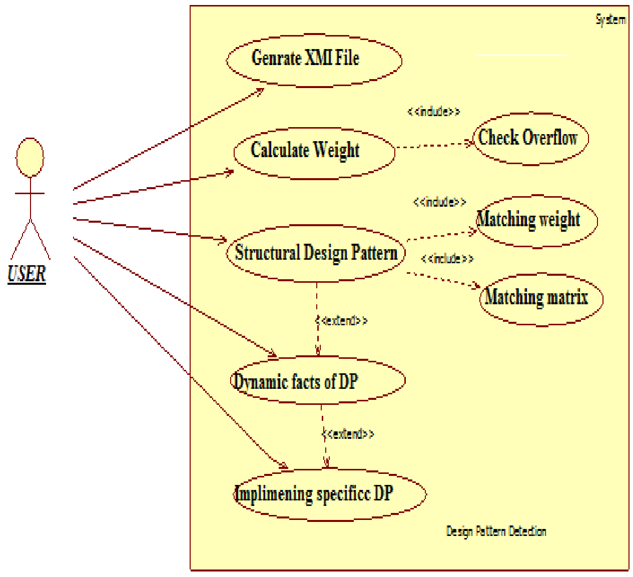


Figure 3.2: Use case diagram

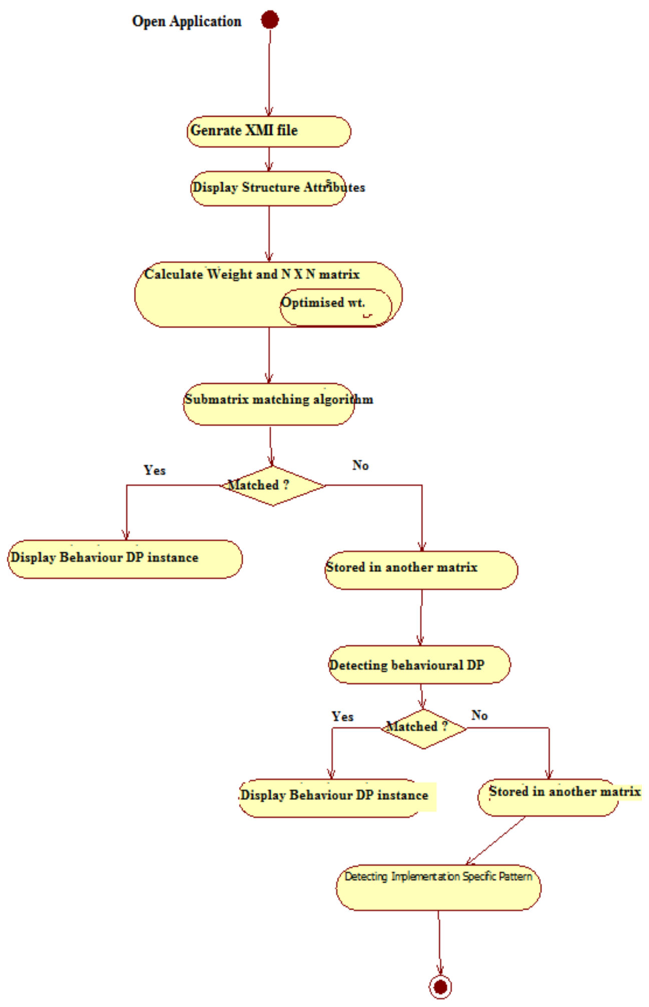

Open Application
Genrate XMI file
Display Structure Attributes
Calculate Weight and N X N matrix
Optimised wt.
Submatrix matching algorithm
Yes
Matched ?
No
Display Behaviour DP instance
Stored in another matrix
Detecting behavioural DP
Yes
Matched ?
No
Display Behaviour DP instance
Stored in another matrix
Detecting Implementation Specific Pattern

Figure 3.3: Activity diagram

C. Steps Required for Design Pattern Detection and Sub Matrix Matching Algorithm:

- Generation of XML/XMI file from java source code through some UML design tool.
- Extraction and `Calculation of structural information's from XML file through SD Metrics tool.
- Calculation of weights of all classes used in java source code and generation of $n \times n$ system matrix and stored them as sparse when required.
- Optimization of weights to prevent overflow problem.
- Candidate design pattern are also encoded into another $m \times m$ matrix.
- Matching of optimized weights of n- dimensional system matrix with m-dimensional candidate design pattern.
- Also matching of candidate design pattern relation matrix (i.e. association, generalization) with relation matrix of system matrix using binary sub matrix matching algorithm.

In this way, the system design information is represented into a two dimensional n × n data matrix where n is the cardinality of classes in the system. Similarly, the information in a candidate design pattern is also represented into another m × m matrix where m is the cardinality of involved classes in the design pattern. If the system design matrix contains so many unchanged entries we stored the n × n and m × m matrix and as sparse and apply sparse matrix algorithm. The identification of candidate design pattern is therefore reduced into the matching of the two matrices.

Once we calculated the optimized weight, association and generalization binary matrix from the XMI file of the java code we can examine whether a particular class satisfies the requirements of a candidate design pattern by matching the optimized weight and binary matrix of a design pattern with those of a system design. If the optimized weights and binary matrix of structured classes of software system is integral multiples of those of the respective classes of a candidate design pattern, this group of interacting classes is acknowledged as a candidate instance of the design pattern.

Besides weight and matrix, we check class type, i.e., if it is an interface, an abstract or a concrete class. Some design patterns may require their participating classes to be of certain types. If we can find a group of such classes, each of which satisfies a particular role of a design pattern, we record them as an instance of that design pattern. Modules that generate data should be separated from a module that consumes data to increase modifiability of system because changes are often confines to either side.

To generate the 2D matrix for a system design and a design pattern we follow the following rules:

- Initially the matrix is represents as $n \times n$ where n is the number of classes involved. Each row and column represents a class arranged in the symmetric order, i.e., row i and column i must have the same class name. Each cell initially has value 1.
- Multiply the value of cell (i, j) by prime number 5 if each class i and class j has association relationship with each other.
- Multiply the value of cell (i, j) by prime number 7 if each class i and class j has generalization relationship with each other.
- Multiply the value of cell (i, j) by prime number 11, if each class i and class j has dependency relationship with each other.
- Multiply the value of cell (i, j) by prime number 13, if each class i and class j has dependency relationship with each other.

TABLE1:

PRIME NUMBERS OF STRUCTURAL ELEMENTS

Structural Elements	Prime Number Value
Attribute	2
Method	3
Association	5
Generalization	7
Dependency	11
Aggregation	13

Formula for calculation of weight of a class is:

W= wa × wm × was × w g × w d × wag

wa = 2(number of attributes in the class)

wm = 3(number of methods in the class)

was = 5(number of Association relationship of the class)

wg= 7(number of generalization relationship of the class)

wd = 11(number of dependency relationship of the class)

wag= 13(number of aggregation relationship of the class)

As from table we assign lower prime numbers to attributes, method since in any class there is more number of attributes and methods. If we assign more prime value then problem of overflow may occur very often. To avoid such overflow we will associate low prime number to more structural element in class.

4. ILLUSTRATION

Here we illustrate the Generalization and Association matrix of bridge design pattern

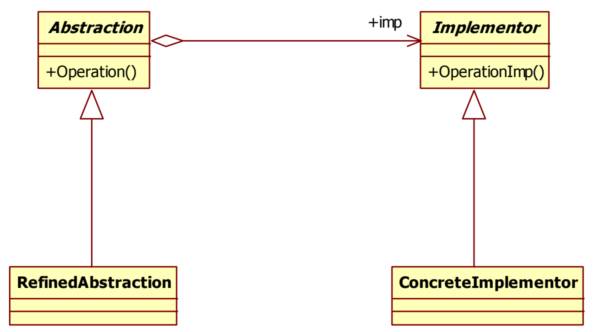
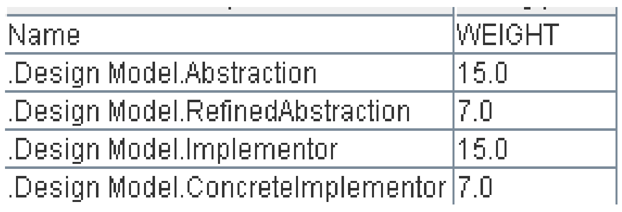


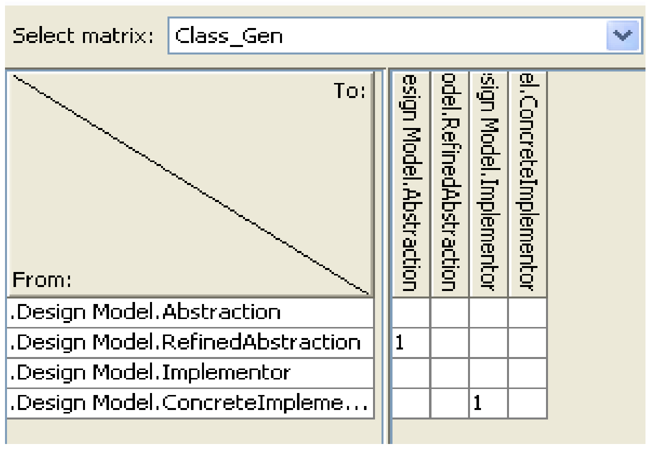
Figure 4.1: Class diagram of bridge design pattern



Name	WEIGHT
.Design Model.Abstraction	15.0
.Design Model.RefinedAbstraction	7.0
.Design Model.Implementor	15.0
.Design Model.ConcreteImplementor	7.0

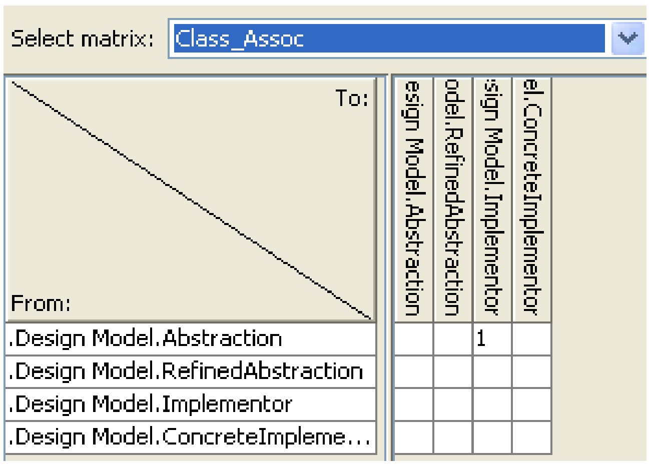
TABLE 4.1:

CLASSES AND ITS CALCULATED WEIGHT USING SD METRICS TOOL



Select matrix: Class_Gen

To: / From:	esign Model.Abstraction	odel.RefinedAbstraction	sign Model.Implementor	el.ConcreteImplementor
.Design Model.Abstraction				
.Design Model.RefinedAbstraction	1			
.Design Model.Implementor				
.Design Model.ConcreteImpleme...			1	



Select matrix: Class_Assoc

To: / From:	esign Model.Abstraction	odel.RefinedAbstraction	sign Model.Implementor	el.ConcreteImplementor
.Design Model.Abstraction			1	
.Design Model.RefinedAbstraction				
.Design Model.Implementor				
.Design Model.ConcreteImpleme...				

Similarly we can generate the association, generalization matrix of all the existing design pattern.

5. RESULT AND FUTURE SCOPE

In this dissertation, we have proposed the new architecture of Automatic Detection of Software design pattern for reverse engineering from java source code, along with partial implementation of the system by converting the java source into higher level of abstraction. The design is quite general, modifiable and flexible, so that it can be merged with similar other legacy systems. The matrix based approach along with classification of design pattern based on reverse taxonomy shows good results in terms of reduced search space and complexity of detection process. Our generated solution is more accurate and effective since I have used the brute force search method for binary sub matrix matching of design pattern from system matrix generated from java source code along with matching of weight and its matrix. The advantage of brute force or exhaustive search method is that it simple to implement and always shows the solutions if it exists.

This project is helpful in the mining of design patterns purpose in the following manner:

- Contribution to area of software maintenance of existing program, component and legacy system.
- Solves the specific design problem of existing legacy software and render conventional and object oriented design better, flexible, elegant and generally extendable.
- It also helps for novices to learn about the functionality, object interaction, and to understand the good object oriented design-intend of the pre-developed software.

There are many issues that requires further attention to resolved them amongst them are to find out suitable architecture and method to detect the rest of the patterns based on reverse engineering i.e. dynamic behaviour and implementation-specific. Space and run time complexity can be further reduce if we apply chain code algorithm for binary sub matrix matching can be applied instead of brute force method. It is generally difficult to separate aggregation from association in reverse engineering processes from source code since they differ at semantic level.

6. REFERENCES

[1] Charles Rich, Linda M. Wills, "Recognizing a Program's Design: A Graph-Parsing Approach," IEEE Software, vol. 7, no. 1, pp. 82-89, Jan./Feb. 1990, doi:10.1109/52.43053.

[2] Linda Mary Wills, Using Attributed Flow Graph Parsing to Recognize Clichés in Programs In Proceedings of the International Workshop on Graph Grammars and Their Application to Computer Science, 1996.

[3] L. Wills, Automated program recognition by graph parsing, Technical Report 1358, MIT Artificial Intelligence Lab, July 1992, PhD Thesis.

[4] Michael Siff and Thomas Reps**,** Identifying Modules via Concept Analysis, IEEE transaction on software engineering, Vol. 25, No. **6,** 1999, pp 749-768

[5] R. Ferenc, A. beszedes, l. fulop and j. lele, design pattern, mining enhanced by machine learning, 21st ieee, international conference on software maintenance, 2005.

[6] Ozalp Babaoglu, Geoffrey Canright, Andreas Deutsch, Gianni A. Di Caro, Frederick Ducatelle, Luca M. Gambardella, Niloy Ganguly, M Ark Jelasity, Roberto Montemanni, Alberto Montresor and Tore Urnes, design patterns from biology for distributed computing, ACM, pp 1-40, 2006.

[7] Shinpei hayashi, junya katada, ryota sakamoto, takashi kobayashi and motoshi saeki, design pattern detection by using meta patterns, special section on knowledge-based software eengineering, IEICE Trans. Inf. & Syst., Vol.E91–D, No.4 April 2008

[8] Jing Dong, Yongtao Sun and Yajing Zhao, Design pattern detection by template matching, Proceedings of the 2008 ACM symposium on Applied computing, Pages 765-769, 2008

[9] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. Halkidis, Design Pattern Detection Using Similarity Scoring, IEEE transaction on software engineering, 32(11), 2006.

[10] V.D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. Van Dooren, A Measure of Similarity between Graph Vertices: Applications to Synonym Extraction and Web Searching, SIAM Rev., vol. 46, no. 4, pp. 647-666, 2004.

[11] J.M. Kleinberg, Authoritative Sources in a Hyperlinked Environment, J. ACM, vol.46, no. 5, pp. 604-632, Sept. 1999.

[12] Niklas Pettersson and Welf Lowe, A Non-conservative Approach to Software Pattern Detection, 15th IEEE International Conference on Program Comprehension (ICPC'07), IEEE Computer Society, 2007

[13] Christian Kramer and Lutz Prechelt, Design Recovery by Automated Search for Structural Design Patterns in Object-Oriented Software, Proc. Working Conf. on Reverse Engineering IEEE CS press, Monterey, November 1996.

[14] G. Antoniol, R. Fiutem and L. Cristoforetti, Design Pattern Recovery in Object-Oriented Software, Program Comprehension, IWPC '98. Proceedings., 6th International Workshop on, 153-160, 1998

[15] G. Antoniol, R. Fiutem and L. Cristoforetti, Using Metrics to Identify Design Patterns in Object-Oriented Software, IEEE Computer Society, 1998.

[16] F. Shull, W. L. Melo, and V. R. Basili. An inductive method for discovering design patterns from objectoriented software systems. Technical report, University of Maryland, Computer Science Department, College Park, MD, 20742 USA, Oct 1996.

[17] K. Kontogiannis, R. De Mori, R. Bernstein, M. Galler, and Ettore Merlo. Pattern matching for clone and concept detection. Journal of Automated Software Engineering, March 1996.

[18] Federico Bergenti and Agostino Poggi, Improving UML Designs Using Automatic Design Pattern Detection, In Proc. 12th. International Conference on Software Engineering and Knowledge Engineering, 2000.

[19] Krzysztof Stencel and Patrycja W egrzynowicz, Detection of Diverse Design Pattern Variants, 15th Asia-Pacific Software Engineering Conference, IEEE Computer Society, 2008.

[20] Jing Dong, Dushyant S. Lad, Yajing Zhao, DP-Miner: Design Pattern Discovery Using Matrix, Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, IEEE Computer Society, 2007.

[21] Francesca Arcelli, Luca Cristina, Enhancing Software Evolution through Design Pattern Detection, Third IEEE Workshop on Software Evolvability, IEEE Computer Society, 2007

[22] Jing Dong, Yajing Zhao, Experiments on Design Pattern Discovery, Third International Workshop on Predictor Models in Software Engineering (PROMISE'07), IEEE Computer Society, 2007.

[23] Damir Kirasic and Danko Basch, Ontology-Based Design Pattern Recognition, Volume 5177/2008, Springer Berlin / Heidelberg, pp 384-393, 2008.

[24] Sven Wenzel, Udo Kelter, Model-Driven Design Pattern Detection Using Difference Calculation.

[25] http://pi.informatik.uni-siegen.de/Mitarbeiter/wenzel/publications/dpd4re06.pdf

[26] Lothar Wendehals and Alessandro Orso, Recognizing Behavioral Patterns at Runtime using Finite Automata, ACM, 2006.