The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.24 e-ISSN: 2251-7545

Design Patterns and Documentation Recovery based on Attributes

Afnan Bashir[1], Ghulam Rasool[2] , Komal Bashir[3], Ayesha Haider Ali[4], Faria Kanwal[5]

*1, COMSATS Institute of Information Technology, Lahore, Email : afnan.bashir@hotmail.com
2, Assistant Professor, Department of Computer Science, COMSATS Institute of Information Technology,
Email : grasool@ciitlahore.edu.pk
3, LCWU, Email : ko_junaid@yahoo.com
4, LCWU, Email : ayesha.iqbal@gmail.com
5, LCWU, Email : faria.kanwal@gmail.com

***Abstract*— The accurate recovery of design patterns from software applications is still debatable and it depends on different types of analysis methods performed on the source code during recovery of patterns. Structural, behavioral and semantic analysis methods are used to extract patterns from source code. Most approaches used combination of these analysis methods to extract patterns from different applications but the recovery process becomes heavyweight. We present a novel design pattern recovery technique based on attributes from .Net applications using only semantic analysis. Implemented attributes enhance the comprehension of source code related with design patterns. A prototyping tool is developed to realize the concept of approach.**

***Keywords*— Design patterns, Reverse engineering, Patterns recovery, Patterns evaluation, Documentation recovery**

I. INTRODUCTION

Design patterns are recurring solutions to standard software problems and they have been used in different applications such as security, web, services, architectures, user interfaces etc [12]. Due to continuous evolutions in software applications, the original design structure is being altered very often. Up gradations of source code is mostly not followed by modifications in the supporting documentation. If developers are given the responsibility to keep source code and documents consistent, this would require extra efforts from developers which results in decreased productivity.

The recovery of design patterns can provide strong indications about the rationale behind the system's design and it helps in the reverse engineering, refactoring and maintenance domains. Each pattern solves a design problem that occurs in software development applications. Mostly design documents are obsolete or missing in legacy systems. Even if the documents are available, they may not match exactly to the source code that may have been changed over time. Due to missing information about the design patterns in source code the restructuring and maintenance becomes arduous.

In maintenance and up-gradation phases the developers require information about the existing structure of an application and implemented design practices. The comments in the source code and documentation can give clues about intention of developers in the source code, but comprehension of complete architecture of a system under analysis is very hard. Maintainability and comprehensibility of an application is directly affected by missing documentation and information about implemented architecture. Lack of information about design patterns implemented in the source code can result in possible incorrect modifications of implemented patterns. This incorrect modification would lead to weakening the structure of an application. The purpose of this paper is to present an approach that facilitate developers in the integration of metadata to source code and recover the implemented design patterns from the source code with the help of embedded metadata. The presented approach is very simple in terms of implementation that it does not cause extra burden on developers.

Throughout entire software design recovery research era, many recovery techniques have been developed and few [6, 9, 10, 11] suggested the use of annotations/meta-information in the source code. Embedding of meta-information in the source code requires extra efforts from developers. The key concerns for an acceptable approach based on meta-information are following:

- How to integrate semantic information in the source code in such a way that it does not require extra efforts from developers?
- Recovery of necessary documentation within the code with recovery of patterns.
- Standardization of meta-information for each pattern.
- Automatic embedding of meta-information in the source code.
- The accuracy in recognized design patterns instances.

International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.24 e-ISSN: 2251-7545

To address the above mentioned issues many techniques [1, 2, 5, 6, 7, 8, 9, 10, 11, 16, 19] have been presented in the past. We want to clarify that we included most relevant design pattern recovery techniques which focused on embedding meta-information during forward engineering of applications in this paper. The techniques along with the tools that are used to recover the design patterns vary across development platforms as well as the variations in design pattern implementations.

Without any tool support the detection of instances of a pattern and relation among the roles present in the source code is very difficult. A developer must be provided with a good presentation of all possible participants of each pattern's instances present in source code. Moreover, due to the lack of ability in identification of participants of a particular pattern it becomes almost impossible to prevent that code block from being altered in such a way that resultant code block would lose pattern integrity and benefits that were gained by the implementation of design pattern are, lost subjecting system to weak architecture [6]. We focus on integration of meta information related with design patterns in the source code which keep source code and documentation of design patterns consistent and enhance the comprehension of source code. The rest of paper is organized as follows:

The section II discusses related work on design pattern recovery approaches which are very similar to presented approach. Section III presents the concept, architecture and detailed implementation of proposed approach. Section IV provides insights about the prototyping tool developed as proof of concept for presented approach. The evaluation of approach is discussed in section V. Section VI highlights the significance and limitations of presented approach. Conclusion and future extensions are discussed in section VII.

II. RELATED WORK

Meffert [11] suggested the implementation of annotations to aid the process of metadata integration into source code fragments. The applied approach aids developers in the selection of the appropriate design pattern implemented. In order to specify the implementation reasoning of some particular source code fragment, the author introduced the usage of annotations. The author emphasize that intents are useful for the support of design patterns implementations; however author did not made any attempt to recover or validate possible pattern implementations within the source code.

Sabo et al. [10] suggested an approach that helped preserving the original structure of implemented design patterns during evolution of an application. The suggested approach aims to separate the intent of the implemented patterns participants in the source code by the use of annotations. The approach also helps in determination of the validity of the applied pattern after the regular maintenance phase. We got inspiration from this work and implemented attributes related with design patterns in the source code.

Rasool et al [6] presented design pattern recovery approach based on annotations in the source code. Authors implemented multiple searching techniques such as SQL queries, source code parsers and regular expressions in the prototyping tool. They developed an add-in for "Enterprise Architect" and extracted structural information stored in SQL database. In this approach authors also focused on recovering variants of a pattern implementation. The recovery process was composed of the structural information as input to the source code parsing engines. Source code parsers were used for detail analysis of source code. In order to reduce search space and increase the probability of detection authors introduced custom annotations embedded within source code in such a way that they integrate crucial information.

Kajsa et al [9] presented an approach for design pattern support based on annotations and feature models. Authors purposed annotations for different design patterns. The author presented aliases for the distinction between same design patterns implementations according to their intent. The authors claim that the presented approach is also capable of handling the variants of design patterns. The implementation of approach is done on Eclipse Framework and templates are developed in JET framework. However, Java does not allow the annotation with identical name which is a major limitation of this approach.

III. PATTERN RECOVERY APPROACH

Various authors [6, 9, 10, 11] recommend the implementation of attributes or annotations for the purpose of embedding semantic metadata to source code fragments. This information can facilitate in design pattern recovery and documentation very amicably. Our approach suggests the use of custom attributes according to each design pattern. Following a particular standard these custom attributes can be developed by a developer himself or can be selected from provided library which is developed to reduce the development time.

The presented approach is capable of identifying the roles that each class plays in multiple pattern implementations. A participant can have multiple attributes implementations according to each role which make our approach capable to recognize overlapping roles of different classes in multiple design patterns. The approach is divided into three key phases which are explained below:

A) IMPLEMENTATION PHASE

The major purpose of this phase is embedding metadata information necessary to participants with the help of custom attributes provided in Spice Library. The implementation phase utilizes the pre-defined attributes from spice library. These attributes facilitate developers to implement the custom attributes defined according to standard GoF patterns specifications. These attributes are manually added on participants of design patterns implemented in the source code. The result of attributes implementation is a source code with embedded metadata information. This code can be then compiled to obtain binaries with metadata integration. During the compilation process metadata information is not lost.

The embedded semantics aid the retrieval of metadata in design pattern recovery process. The suggested custom attributes can be applied on all existing Gang of Four design

International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.24 e-ISSN: 2251-7545

patterns as well as on custom design practices. In order to prevent chances of conflicts with the pre-defined attributes implemented in .NET Framework custom attributes can be used. Attribute [17] is the base class in .NET Framework and it has a very powerful feature that it allows developers to extend it and create their custom Type [18].

Custom attributes facilitate developers in implementation of attributes other than pre-defined attributes. For the suggested approach it is necessary to create custom attributes in order to increase accuracy and to reduce any possible conflicts with pre-defined system attributes that may hinder in design recovery phase in later stages.

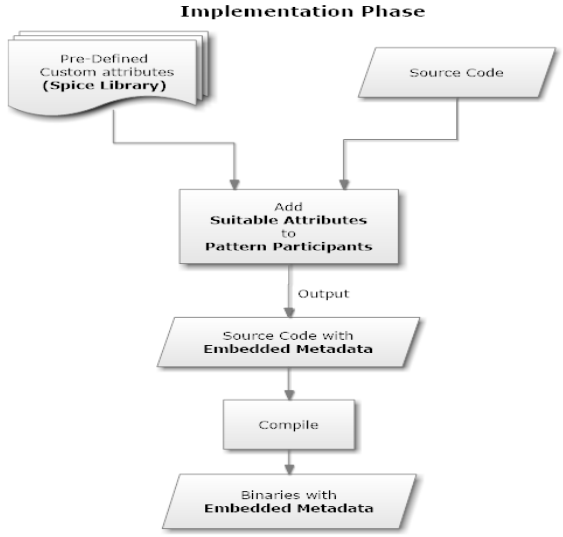

Fig. 1 Overview of Implementation phase

The above Fig1 illustrates the implementation phase in detail. The implementation phase is carried out manually during the time of application development. The following Spice library is key component of this phase.

1) Spice Library

In order to create and implement the custom attributes for each design pattern, extra burden comes on developers. This can cause trouble achieving milestones within specified time period. To reduce the additional development burden added by this suggested approach we developed the Spice Library. Spice library contains pre-defined custom attributes developed considering each pattern present in the Gang of Four patterns.

The custom attributes defined in Spice Library are based on rules that help the recovery process. Developers can create their custom attributes keeping the rules under consideration. The implementation of attributes suggested in Spice Library improves the comprehension of source code. Custom attributes suggested in Spice Library act as bucket that collects all the vital information necessary for design pattern recovery as well as the intent of implementation of that particular design pattern code fragment. The example of custom attributes on singleton design pattern is explained as follows:

Custom attributes implemented on client

```
[SpiceLibrary.Creational_Patterns.Singleton_Pattern.Client(3,
"Client Load Balancer", "Load Handlers")]
class worker
    {
        public void work()
        {
            //listen to server1
            LoadBalancer b1 = LoadBalancer.GetLoadBalancer();
            //listen to server2
            LoadBalancer b2 = LoadBalancer.GetLoadBalancer();
        }
    }
```

Custom attributes implemented on *Base Class*

```
[SpiceLibrary.Creational_Patterns.Singleton_Pattern.Singlet
on(3, "Used to connect to server it should be same instant
for all requests", "Base")]
    public class LoadBalancer
    {
        private static readonly LoadBalancer _instance =
new LoadBalancer();
        // Type-safe generic list of servers
        private List<Server> _servers;
        private Random _random = new Random();
        private LoadBalancer()
        {
            _servers = new List<Server>
                {
                 new Server{ Name = "ServerI", IP =
"120.14.220.18" },
                 new Server{ Name = "ServerII", IP =
"120.14.220.19" },
                 new Server{ Name = "ServerIII", IP =
"120.14.220.20" },
                 new Server{ Name = "ServerIV", IP =
"120.14.220.21" },
                 new Server{ Name = "ServerV", IP =
"120.14.220.22" },
                };
        }
        public static LoadBalancer GetLoadBalancer()
        {
            return _instance;
        }
        public Server NextServer
        {
            get
            {
                int r = _random.Next(_servers.Count);
                return _servers[r];
            }
        }
    }
```

B) EXTRACTION PHASE

This phase targets the extraction of possible design patterns implemented in the source code. It accepts binaries of application on which the annotations were implemented with proper metadata information. Extraction phase takes into account the principles of Reflection [13]. Reflection is very powerful feature introduced by Microsoft in .Net framework. Reflection [13] allows you to retrieve information about the assembly that may be an executable or dynamic link library. It can help extracting metadata related to classes, interfaces and value types. Reflection is useful for both static and dynamic analysis for design pattern recovery process. It is not necessary to perform implementation phase each time. Fig 2 provides the overview of the extraction phase.

To extract all the custom attributes declared in source code the assembly is loaded into reflection object. The reflection object loads the assembly and extracts

International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.24 e-ISSN: 2251-7545

the metadata. The metadata of attributes remains intact even after compilation. Once Custom attributes are extracted they are analyzed to filter out the possible pattern's candidates. These are further sorted to arrange attributes according to pattern category and fed into pre-computation engine that extracts necessary information about attributes. The pre-computed data and list of organized pattern attributes is fed into "pattern analysis and recognition" engine to extract and validate the possible instances of a pattern.

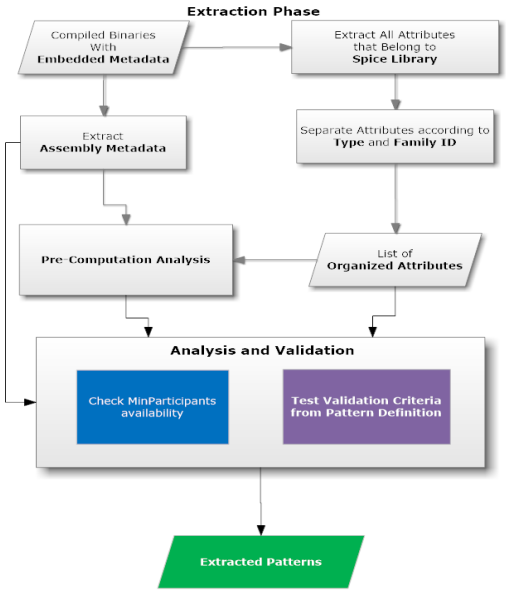

Fig.2 Overview of Extraction Phase

C) Pattern Analysis and Recognition Engine

This phase is most important during pattern recovery process. It accepts the input (sorted according to type and family ID) from first phase of extraction process, where all the possible pattern candidates were extracted from assembly. The core purpose of this engine is to take all possible candidates and evaluate their legitimacy. This helps to decide that either selected candidate is a valid pattern implementation, a misplacement of attributes or in-accurately implemented design pattern. This engine performs analysis according to given functional definition. The recognition engine is responsible for the detection, extraction and validation of design patterns present in an application. This engine consists of two major phases discussed as follows:

1) Pre-Analysis Computations

In order to optimize the analysis and validation phase some computations are performed prior to analysis phase. Pre-analysis phase involves the extraction of critical information which would help reducing the extra efforts obtaining such data during analysis process. The applied technique helps reduction of complexity while increasing the efficiency and results of process. This information is crucial for analysis. The pre-analysis calculates following information:

- Number of methods present in that namespace
- Family ID of a pattern candidate that identifies its group in same category
- Title of a pattern candidate
- Comments included in a pattern candidate
- Name of a Loaded Assembly
- With the help of Reflection extract name of class in which participants of a pattern exists.
- Get name of a possible pattern. This is decided on basis of implemented attribute's structure. This is for reducing search space and it is validated in later phase.
- Get names of all participants

The above extracted data is stored in a custom type called CodeData and is used for analysis and validation.

2) Analysis and Validation Phase

This phase is responsible for determining the legitimacy of a candidate pattern. Provided with the information from pre-analysis phase, this phase utilizes that information to extract relationships among classes, interfaces, delegation and class inheritance. This involves validation of a candidate according to particular design pattern function definition pre-defined within the engine. Writing a custom function definition requires skills and information about minimum set of rules that should be met before it is considered a valid pattern implementation. Function definitions require different set of instructions. Fig 3 presents the flow chart diagram for analysis and validation of a proxy pattern.

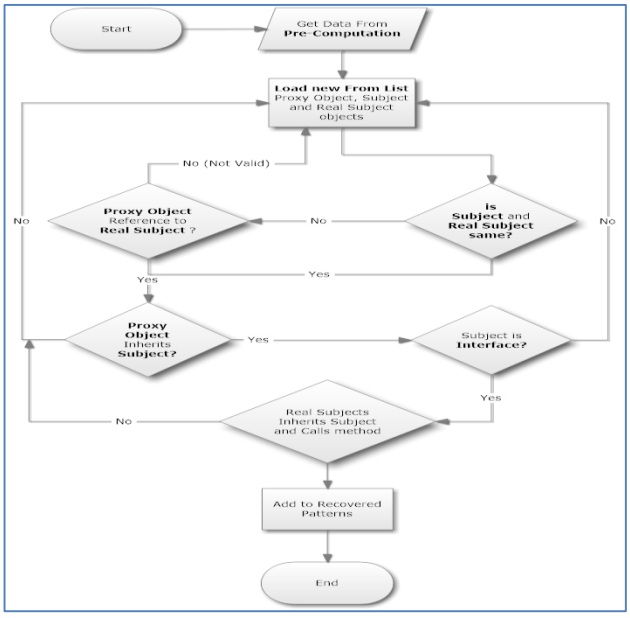

Fig3. Flow diagram for proxy pattern detection

IV. Prototyping Tool

A prototyping tool called SPE (Spice Pattern Extractor) was developed as a proof of concept for the suggested approach. The current implementation of tool focuses on patterns belonging to creational and structural categories of

International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.24 e-ISSN: 2251-7545

the GoF patterns. Most design pattern recovery techniques are supported by different tools for validation of their results. Each tool is developed with a particular methodology and it is very difficult to integrate one tool with another. Most tools designed for design pattern recovery have very little or even no support for the process of documenting the existence and usage of patterns present in the source code. The tools differ by various aspects i.e. algorithms, pattern descriptions, pattern representation, precision, recall etc. The following are points that have been framed for our prototype tool:

1) The output presentation of recovered patterns should facilitate source code comprehension process.
2) Tool should be flexible and can be extended without problems.
3) Modularity should be in such a way that modification in one module does not affect other modules.
4) Tool should be able to recover patterns only on the basis of semantic analysis.

A. *The Concept and Architecture*

The tool utilizes the metadata for the recovery of design patterns. SPE does not use source code parsers to extract information for analysis. It performs analysis in three steps: extraction, analysis and validation as discussed in previous section. Extraction reads the binaries and extracts the basic structure of participants on which attributes are applied. The extracted data ensures the availability of necessary structural information required by analysis step. The analysis step utilizes the information provided from previous step to get the minimum participants that a pattern implementation should have. This information is contained in Spice Library and is embedded during the time of attribute declaration. This step filters out the patterns which satisfy the minimum participants condition. Accepted patterns are stored in a list of custom structure. Validation is the process of verifying certain rules that a pattern should meet before it can be declared a valid pattern implementation.

B. *Flexibility and Customization*

One of the major considerations during the development of SPE was modularity. SPE is based on various modules as discussed in section VI and illustrated in figure 4. The benefit of using modular design is the ability to modify a module without effecting other modules or components. SPE is highly customizable and user with basic programming skills set can create and implement function definitions that can help in recognition of custom design practices. SPE also allows user to fine tune the function definitions and custom attributes implemented in spice library. Custom function definitions can extend the ability to detect broader range of patterns.

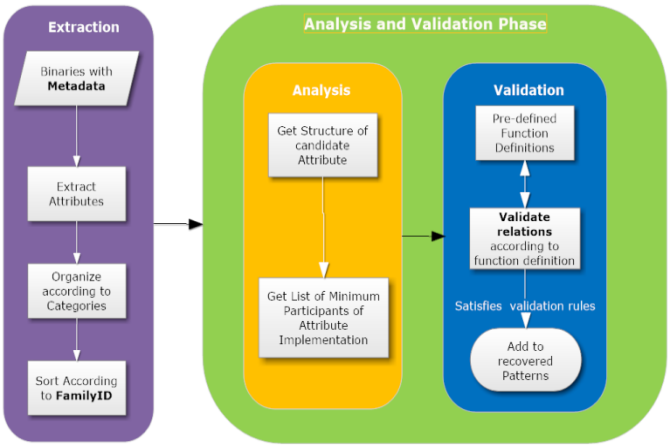

Fig.4 Overview of Tool Architecture

C. *Accuracy and Experimentation*

A good recovery tool should be able to match and extract the required design patterns accurately with high precision and recall rate. The precision and recall metrics help in the evaluation of information extraction techniques including design pattern recovery approaches. Accuracy of an approach is determined by precision and recall metrics. SPE was tested on an open source LAN messenger application to extract the design patterns implemented in this application. Prior to extraction source code was implemented with the custom attributes from Spice Library for the purpose of embedding necessary metadata information. During the recovery process all design patterns were successfully recovered on which attributes were applied. The tool recovered patterns with good precision and accuracy however it is necessary to compare SPE with some other commercial tools and revalidate the results obtained by SPE on few other commercial and industrial applications. Due to lack of trusted benchmarks the results obtained after the analysis were manually analyzed.

V. Evaluation of Approach

Validation is very important step for analyzing worth and performance of purposed approach. This paper has introduced a novel approach for design patterns recovery and nothing relative has been done in the past to the best of our knowledge. Our approach relies on attributes for the detection of design patterns. Function definitions help the tool in the validation of the recovered pattern's candidates to confirm their legitimacy. We have focused on developing pattern recovery process to recover patterns from C# applications. The scalability of our suggested approach is validated by the experiments we have performed on an open source application named "Squiggle" [14]. Due to absence of trusted benchmarks the comparison with any other approaches was not possible. The results obtained by SPE were manually validated.

Considering ourselves as pioneers of the presented approach, we adapted the manual detection and validation of source code to compare results obtained from tool. We spent long hours to review source code manually and implemented suitable attributes from Spice Library on the analyzed patterns. The source code was then compiled to obtain the

International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.24 e-ISSN: 2251-7545

binaries which were later on provided to SPE for detection of implemented design patterns.

The results obtained were compared with the patterns on which attributes were applied. SPE did not miss any instance in examined application; however there is a need for the experimentation on different commercial and industrial applications. Our suggested pattern recognition approach extracted the implanted design patterns and other artifacts participating in a pattern implementation.

A. *Results*

Precision and Recall are two very important metrics for the evaluation of all design pattern detection approaches [4]. Precision and recall have been part of quality assurance of systems for a long time. The accuracy of an approach is determined by the relationship between precision and recall [4]. In ideal conditions precision is directly proportional to recall but these ideal conditions are hard to achieve [21]. Both precision and recall are heavily dependent on the nature of design pattern recovery approach implemented in a tool. Table1 presents the results of our prototyping tool.

TABLE 1
RECOVERED DESIGN PATTERNS INSTANCES

Pattern Name	Attribute Implemented	Detected by SPE
Creational Patterns		
Abstract Factory	0	0
Builder	0	0
Factory Method	0	0
Prototype	**2**	**2**
Singleton	**1**	**1**
Structural Patterns		
Adapter	**1**	**1**
Bridge	0	0
Composite	0	0
Decorator	0	0
Façade	0	0
Flyweight	0	0
Proxy	**1**	**1**

The Precision and recall metrics are calculated by true positives, true negatives, false positives and false negatives [4].Precision and recall are 100 % which yields F-Score of 100%. The only option to validate our approach was to populate our own benchmark by analyzing source code manually.

VI. Significance and Limitations of Approach

The presented approach is initial step in software design pattern recovery from .Net applications. The patterns are recovered only on basis of semantic analysis. No source code parsing had been performed and yet results are very precise as discussed in previous section.

Following are few major points that make our approach much vibrant.

- No source code parsers are required for the extraction of design patterns.
- Enhanced source code quality by addition of attributes/annotations which helped improvement in comprehension of code.
- Metadata and semantics were present as object. This helps to achieve fast, robust and accurate solution for design patter recovery.
- Allows user to extend their design implementations beyond regular GOF patterns.
- Provides insights about the location of design patterns existence and their participants in various different classes or interfaces with their metadata information.

Design pattern recovery approaches cannot be generalized and one solution cannot fit all problems. Our suggested approach contributes to the problems like accuracy, flexibility, precision and extensibility in design pattern recovery, but on the other hand this approach is subject to some limitations. Following are the plausible limitations of suggested approach:

- Approach is highly dependent on reference that one binary contains. If any reference is lost then recovery process may not yield proper results.
- The implemented attributes should follow particular format as implemented in Spice Library.
- This approach only recovers patterns from the source code in which defined attributes are implemented.

VII. Conclusions Future Work

The main purpose of this paper is recovery of design patterns and related documentation from C# applications based on attributes. The presented approach reflects modularity which makes it a sustaining approach against all future challenges. The applied approach utilizes the metadata embedded in source code with the help of attributes provided in Spice Library for extraction of patterns. The approach is easily customizable that it can handle variants of design patterns. The experiments are performed on open source library squiggle[14] by using our prototyping tool SPE. Due to the absence of trusted benchmarks related to approach, the validation of results was a challenge. Validation is performed by calculating precision, recall and F-Score. The results obtained are remarkably accurate. We plan to extend approach on all types of GOF and other types of patterns in future. The prototyping tool will be used as add-in with Sparx Systems Enterprise Architect Modeling tool[20] to support the visualization of recovered results. Finally, approach will be evaluated from academia and industry for its efficiency and effectiveness.

The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3n3.24 e-ISSN: 2251-7545

REFERENCES

[1]. C. Kramer and L. Prechelt, "Design recovery by automated search for structural design patterns in object oriented software", in Third Working Conference on Reverse Engineering, Amsterdam. March 1996, pp.208-215
[2]. G. Antoniol, R. Fiutem and L. Cristoforetti, "Design Pattern Recovery in Object-Oriented Software", in 6th International Workshop on Program Comprehension. 1998. pp.153-160.
[3]. K. Kontogiannis, R. De Mori, R. Bernstein, M. Galler, and Ettore Merlo, "Pattern matching for clone and concept detection", in Journal of Automated Software Engineering. 1996. pp.150-155.
[4]. Ghulam Rasool, Detlef Streitfdert, "A Survey on Design Pattern Recovery Techniques", in IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 6, 2011, pp. 251-260.
[5]. J. Dong, J. Zhao, and Y. Sun, "A Matrix based Approach to Recovering Design Patterns", in IEEE transactions on Systems, Man and Cybernatics, Vol 39. Nov 6, 2009. pp. 1271-1282
[6]. G. Rasool, I. Philippow, P. Mader, "Design Pattern Recovery Based on Annotations", in International Journal of advances in Engineering Software, Vol 41, Issue 4.2010. pp. 519-526
[7]. Adrian Paschke, ―A Semantic Design Pattern Language for Complex Event Processing‖, Association of Advancement in Artificial Intelligence (www.aaai.org), 2009, pp. 54-60
[8]. Awny Alnusair and Tian Zhao, Towards a Model-driven Approach for Reverse Engineering Design Patterns‖, In Proceedings of the 2nd International Workshop on Transforming and Weaving Ontologies in Model Driven Engineering (TWOMDE 2009), Denver, Colorado, USA, October 4, 2009, pp. 1-15
[9]. Peter Kajsa , Pavol Návrat, "Design Pattern Support Based on the Source Code Annotations and Feature Models", in SOFSEM 2012: Theory and Practice of Computer Science Volume 7147, 2012, pp 467-478.
[10]. Sabo, M., Porubän, J, "Preserving Design Patterns using Source Code Annotations", in Journal of Computer Science and Control Systems,2009, pp.53–56
[11]. Meffert, K, "Supporting Design Patterns with Annotations", in Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based System, ECBS 2006. IEEE Computer Society, Washington, DC (2006) , pp. 437–445
[12]. Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides, "Design Patterns: Elements of Reusable Object Oriented Software", Addison-Wesley Publishing Company, Reading, MA, 1995, ISBN: 0201633612.
[13]. Reflection. http://msdn.microsoft.com/library/system.reflection.aspx
[14]. Squiggle (Open Source LAN Messenger). http://squiggle.codeplex.com/
[15]. Object. http://msdn.microsoft.com/en-us/library/system.object.aspx
[16]. Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur and Yarden Kartz, ―A Practical OWL-DL Reasoner, In Journal of Web Semantics: Science, Services and agents on the World Wide Web, Volume 5, Issue 2, 2007, pp. 51-53
[17]. Attribute. http://msdn.microsoft.com/en-us/library/e8kc3626.aspx
[18]. Type. http://msdn.microsoft.com/en-us/library/system.type.aspx
[19]. K. Meffert, I. Philippow "Supporting Program Comprehension for Refactoring Operations with Annotations", In Proceedings of the 2006 conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the fifth SoMeT_06, 2006, pp.48-67
[20]. Sparx Systems Enterprise Architect: http://*sparxsystems.com/*
[21]. Scientific Tool works Inc. Understand for C++, 2003, http://www.scitools.com/