A Survey on Design Pattern Recovery Techniques

Ghulam Rasool[1], and Detlef Streitfdert[2]

[1] Computer Science Department, Comsats Institute of IT Lahore, Pakistan
Lahore, Punjab 54000, Pakistan

[2] Software Systems/Process Informatics, TU Ilmenau
Ilmenau, Thüringen 98693, Germany

Abstract

The evaluation of design pattern recovery techniques and tools is significant as numbers of emergent techniques are presented and used in the past to recover patterns from source code of legacy applications. The problem of very diverse precision and recall values extracted by different pattern recovery techniques and tools on the same examined applications is not investigated thoroughly. It is very desirable to compare features of existing techniques as abundance of techniques supplemented with different tools has been presented in the last decade. We believe that new innovations for this discipline can be based on the empirical evaluation of existing techniques. The selected techniques cover the whole spectrum of state of the art research in design pattern recovery. The major contribution of this paper is a comprehensive discussion on state of the art in design pattern recovery research in the last decade followed by a proposed framework for classification and evaluation of existing design pattern recovery techniques. Finally we listed our observations as lessons learned which hamper design pattern recovery research and these observations can be used for future research directions and guidelines for this discipline.

Keywords: *Design patterns, Reverse engineering, Patterns recovery, Pattern evaluation, Empirical studies*

1. Introduction

The field of design pattern recovery has become mature enough in the last decade due to new innovations and a number of rather provoking presentations [1 2 3 4 5 6 7 8 9 10], but it still faces a number of key challenges. The central objective of pattern recovery approaches is to accurately detect patterns from the source code which facilitates software maintenance, program comprehension, refactoring, restructuring, reverse engineering and reengineering disciplines. New patterns have been developed which are used in different areas such as software architectures, user interfaces, concurrency, security and services etc. Developers adopting patterns and practices can expect an average productivity increase of 25 to 40 percent, depending on their skill level and complexity of application [32]. The study [18] revealed that the information extracted through design patterns is very important during the maintenance of legacy applications. Furthermore, the reverse engineering of patterns from existing legacy applications and their reusability for developing new applications enable software developers to leverage best practices encapsulated as design patterns.

It is difficult to compare the efficacy and usability of the pattern recovery tools in an accurate and controlled way, especially when large numbers of tools are developed as research prototypes which support only particular novel methods, but they cannot be generalized. The customization, integration, interoperability, scalability and accuracy of presented tools are important factors while investigating features of different tools. The disparity of results extracted by different techniques and tools motivated us to investigate the causes of disparity in the results. We focus on the evaluation of design pattern recovery techniques and tools with regards to their accuracy and highlight the problems and limitations of existing design pattern recovery techniques.

An empirical review and evaluation of existing techniques is important to guide researchers through the strengths and limitations of existing techniques. The results of evaluation can be used for the adoption of existing techniques and an iterative development of new tools. Pattern recovery techniques and tools differ with respect to the applied analysis techniques (structural, behavioral, semantic or combination of two/three), matching mechanisms (approximate, exact), pattern representations(FOL, ASG, XMI, BPSL etc.), system representations(AST, graph, matrix etc.), accuracy (precision, recall etc.), recovered pattern instances(creational, structural, behavioral etc.), presentation format(textual, graphical etc.), language support(C/C++, Java etc.) and tool support(third party tools, self developed tools etc). The evaluation of pattern recovery techniques becomes difficult when the applied tools are not available publicly for validation of their results. The implementation variants of design patterns has a negative impact on the accuracy of tools and are the major cause of disparity in the

results of different techniques. Furthermore, the problem of scalability has been recognized as an important stumbling point.

The focus of the empirical study presented in this paper is to evaluate what makes any technique different from others as numbers of approaches are presented. The future of design pattern recovery techniques and tools will be based on evidence about the proven effectiveness and usefulness of different techniques and tools. It is useful to gather results of different studies in order to form a large body of knowledge that can be used for the evaluation and adoption of existing techniques and tools. For this purpose we collected, organized and analyzed several sources of information related with pattern recovery techniques. Both quantitative as well as qualitative analysis of pattern recovery instances extracted by different tools is important for the evaluation of tools. Design pattern recovery techniques can be evaluated based on different features, but we focus our analysis specifically on evaluating the precision and recall of different techniques. The motivation for evaluating precision and recall stems from the disparity of results.
The assessment of pattern search algorithms used in different techniques was done based on the following values and terms:

1 A search result is true positive, in case a pattern was found and is really existing in the source code.
2 A search result is true negatives if a pattern is not recognized and not implemented. Of course this is just mentioned for completeness.
3 A search result is false positive, in case a pattern was found, but is not implemented in the source code.
4 A search result is false negative, in case a pattern is implemented in the source code, but was not found.
5 *Precision* – is the ratio of found patterns divided by the number of existing patterns.
6 ***Recall*** – is the number of the implemented patterns divided by the number of found patterns.
7 ***F-Score*** – is the combined effect of precision and recall.
8 ***Accuracy*** – measures that how accurate the results of any technique are.

The paper is organized as follows: Section 2 discusses an overview of pattern recovery techniques and parameters which are important for the evaluation. Section 3 discusses the state of the art selection and statistics of approaches in this area. Section 4 presents our proposed framework which can be used for evaluation. The accuracy of the selected pattern recovery techniques on the same examined systems is discussed in section 5. Section 6 discusses disparity in extracted results. Section 7 presents critical review and our observations. Finally, section 8 concludes with our recommendations and guidelines.

2. Overview of Recovery Techniques

We summarize the overall spectrum of research in the field of design pattern recovery in this section. There are number of factors which can be used for classification of available literature as mentioned in the first section. We classify pattern recovery techniques based on the type of analysis used by the particular technique and the searching methodology adopted.

2.1 Analysis Type

Pattern recovery approaches are classified into structural analysis, behavioral analysis, semantic analysis and formal specification/composition analysis to recover patterns from the source code of different legacy applications.

Structural analysis approaches are based on recovering the structural relationships from different artifacts available in the source code. They focus on recovering structural design patterns such as Adapter, Proxy and Decorator etc. Structural analysis based approaches focus on inter-class relationships to identify the structural properties of patterns, but they completely miss the behavioral aspects. Structural analysis approaches explore the relationships: class inheritance, associations, friend relationships, interface hierarchies, modifiers of classes and methods, method parameters, method return types, attributes and data types etc. Some of the structural analysis approaches extract inter-class relationships from the source code using different third party reverse engineering tools and then perform pattern recognition based on extracted information. For example, reference [14] parses the source code using the third party commercial tool called Understand for C++ [44]. The tool extracts the entities and the references from C++ source code and stores its results in a database. Furthermore, queries are performed on the database to extract different properties of patterns. In [44] the authors recovered Singleton, Factory method, Template method, Observer and Decorator from a VCS (Version Control System). The experiments are performed on a VCS containing only 125 classes that are not available publicly. Thus, all the scalability measures of the approach are questionable.

Behavioral analysis approaches take into account the execution behavior of the program. These approaches are based on dynamic analysis, machine learning and static program analysis techniques to extract behavioral aspects of patterns. They play important role when structural analysis approaches fails to identify patterns accurately that are structurally identical or have a weak structure. These approaches are supplemented by the structural analysis approaches to recover different patterns. For example, State and Strategy patterns are structurally identical. Similarly, Chain of responsibility, Decorator and Proxy have similar structures. The behavioral analysis deals with a small number of classes and gives many *false positives* when the number of execution traces significantly increases. The major difficulty in behavioral analysis is that there may be various possible implementations for the same expected behavior [4]. These approaches face problems of data coverage in the case of large examined applications. The

behavioral analysis techniques narrow down the search space using inter-class relationships and then use (machine learning, dynamic analysis and static program analysis etc.) techniques to extract patterns from a number of legacy applications.

Semantic analysis approaches supplement the structural and behavioral analysis approaches to reduce the *false positive* rate for recognition of different patterns. The semantic analysis approaches [4 5] used the naming conventions and annotations which contain the role information about the classes and methods. Semantic analysis becomes important for recovery of patterns which have similar static and behavior properties. For example, Bridge and Strategy patterns have the similar structural and behavioral characteristics as shown in fig 1. The semantic analysis can be used in such cases. A number of "Gang of Four" patterns are similar in structural and behavioral aspects to a large extent but they only differ in intent for which they are used. The Strategy, State and Bridge are examples of such patterns. Different techniques are used for semantic analysis. In [4], three options are discussed for semantic analysis and we conclude that naming conventions are most appropriate and feasible option.

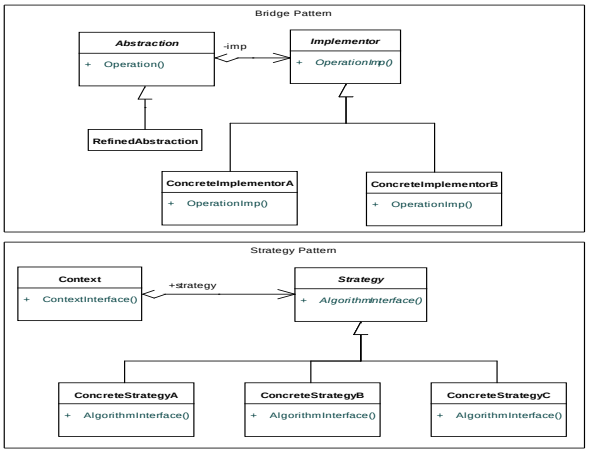

Fig. 1 Bridge and Strategy Patterns.

The formalization of design patterns is another important area that we take into account during our review because some approaches extract patterns from source code based on formal specifications of design patterns. The formal specification [45] of patterns is also important for the composition of different patterns. A number of approaches focused on the formalization and composition of design patterns to supplement different pattern detection approaches by formally specifying the patterns [45 46 47 48]. Formal specification languages are used to specify different design patterns. Some design pattern detection techniques use pattern specifications of other approaches in their implementation to detect patterns based on the source code[5 14 50]. Most of the specification languages have tool support to validate the specifications for correctness and completeness [46]. Semi-formal approaches, such as UML specifications and textual descriptions cannot capture the essence and the intent of patterns. Finally, design patterns have different implementation variants and any formal specification of patterns can help to specify the possible variations in different patterns as well as overcome the challenges of capturing the semantics for patterns.

2.2 Detection Methods

Different pattern recovery techniques are using different methods for searching patterns from the source code. The overview of most important techniques is given below:

Database queries

Database queries are used by number of design pattern recovery techniques [5 8 14 15 23] for extracting patterns. These approaches transform source code into intermediate representations like (ASG, AST, XMI, metadata and UML structures etc.) and then use SQL queries to extract pattern related information from particular representations. The performance of the queries to extract related features of design patterns is directly bound to the database in use and can be scaled very well, but such queries are limited to the information which is available in the intermediate representations. To the best of our knowledge, no intermediate representation format is currently available which could store all the information present in source code. The approaches based on SQL queries are also restrictive to structural and creational design patterns so far and they only partially support behavioral design pattern recovery.

Constraint resolver

The PTIDEJ[26] team developed the Ptidej tool suit, a reverse engineering framework to identify idioms, macro-patterns, design patterns and design defects using explanation based constraints programming technique. It is a very active group, continuously involved in extensive research projects. They have developed different other tools like (DPR, DeMIMA, DÉCOR etc.), and describe design motifs as constraints systems where each role is represented as a variable. Relationships among roles are represented as constraints among variables. The PTIDEJ team recovers patterns using a multilayered approach which focuses on ensuring a 100% recall rate, but precision is scarified and performance is low.

Metrics

Metric based techniques compute program related metrics (generalizations, aggregations, associations, interface hierarchies etc.) from different representations of source code and then use different techniques to compare metric values of each design pattern definition with source code metrics. Metric based techniques are computationally efficient because they reduce search space through filtration [7]. These approaches [24 43 50 51] performed experiments on very few patterns and their generalization for recovery towards all types of the GoF [25] patterns is questionable. In

addition, these approaches are also not interactive and report low precision and recall.

XPG formalism and parsing

These techniques are using the SVG (scalable vector graphics) format for the intermediate representation of the source code and design patterns are represented in a visual language. Patterns are recovered using a visual language parsing technique by mapping the visual language grammar of each pattern with the graph representation. These approaches have the advantage of their visualization, have a good precision, but are limited only to structural design patterns. Authors presented a new approach for recovering behavioral design patterns which include only the Observer, Strategy and State pattern [17]. The authors did not report any recall rates for the examined applications. The pattern instances detected by these approaches have variations with the other approaches [1 2 5] as shown in Table 4.

UML structures and matrices

These techniques [2 4 24] represent structural and behavioral information of software systems as UML structures and matrices. They apply different techniques to match the design pattern template metrics with the matrices generated for the system. These approaches are computationally efficient, have good precision and recall rates, but they are not interactive. They are not capable to extract the implementation variants of similar design patterns. Furthermore, matrix based approaches are only restrictive to few number of patterns and they are not able to recover the complete set of the GoF [25] patterns.

Miscellaneous techniques

The remainder of the well known techniques which are presented in different papers are given in Table 1.

Table 1: Miscellaneous Techniques

Authors	Technique
Niere et al [19]	Fuzzy reasoning
Kaczor et al [7]	Bit vector
Shi and osslon [6]	Data flow and control flow
Philippow et al [10]	Minimum key structure
Bayer et al[12]	Predicate calculus
Smith et al [13]	rho calculus
Heuzeroth et al [21]	Runtime analysis
Blewitt [22]	Formal Semantic
Balanyi et al [20]	XML matching
Wang et al [34]	REQL query
Frenc et al [36]	Machine learning
Huang et al[37]	Structural and behavioral parsing
Park et al [39]	Static Reference Flow analysis
Tonella et al [40]	Concept Analysis
Arceli et al [41]	Data Mining

The empirical evaluation of above mentioned techniques demands that each applied technique should be evaluated on the basis of certain criteria. Different authors have suggested taxonomies and/or frameworks for the evaluation of techniques used in the area of reverse engineering. Such taxonomies can be used by the researchers for the empirical evaluation of studies presented in the past. The definition of common and agreed criteria for evaluation of design pattern recovery techniques requires a major effort from the design pattern research community. We suggest the following parameters to evaluate different techniques which are also partially suggested by different other authors [3 16 30 35].

Input: What type of input (source code, source code language, executable, documentation, tests, and intermediate representations) it accepts.

Output: What output it produces, e.g., textual, visual, diagrams, hypertext etc.

Analysis type: What type of analysis it uses, e.g., structural, behavioral, semantic and combination of these two/three.

Automated/Semi-automated: Is it completely automated (require no user involvement) or is it semi-automated (human involvement for certain steps)?

Variant handling (customization): Does it accept customizable pattern definitions to recognize variations or will it only recognize standard pattern definitions?

Scalability: What is the scalability of an applied tool?

Recovered patterns: Which pattern types it recovers (creational, structural, behavioral, user defined etc.)

Intermediate representations: Which type of intermediate representation it uses?

Experiments: Which case studies are selected as experiments? What is size of case studies? Is source code of these case studies available?

Accuracy: What are the precision, recall and F-score?

Pattern representation: How patterns are represented (formal specification, visual specification etc.)

Matching roles: Which type of matching mechanism is adopted (partial match, exact match etc.)

3. State of the Art Selection

We reviewed and selected 89 papers published in highly ranked journals and conferences in the last decade in the area of design pattern recovery as given in Table 2. A number of selected papers published in IEEE/ACM conference proceedings are also included in our review. We found only one volume of workshop proceedings in the area of design pattern recovery [27] and those published papers are part of our review. We filtered out 16 papers based on our selection criteria which are examined on very small examples and they extracted a few patterns which are relatively easy to detect. 73 papers which directly focus on the topic of design patterns and their recovery are part of our statistical analysis.

Firstly, we classified papers according to their use of different techniques in the area of design pattern recovery as shown in the first section of Table 3 (“Objects of Study”). Objects are the entities under study in empirical investigation [33]. The object of our empirical study is the discipline of design pattern recovery. The large number of

papers aims at pattern recovery and they present their results in different formats (incomplete roles, complete roles etc.) without visualization. A small number of papers focused on visual presentation of extracted results which is very important for the comprehension and maintenance of legacy source code. We also included papers in this category, which focus on design pattern visualization at the design level. The objective of a third category is to formalize design patterns in order to highlight overlapping and compositions among design patterns. A number of techniques used formal definitions of these approaches as input in their implementations. Pattern recovery techniques paid little attention on the recovery of overlapping and compositions in extracted pattern instances.

Table 2: Names and Sources of Selected Proceedings

Journals	**Conferences**
IEEE Transactions on Software Engineering http://www.computer.org/portal/web/tse/	International Conference on Software Engineering(ICSE) http://www.icse-conferences.org/
ACM Transactions on Software Engineering http://tosem.acm.org/	Automated Software Engineering (ASE) http://ase-conferences.org/
Journal of System and Software http://www.elsevier.com/locate/jss	Software Engineering and Knowledge Engineering(SEKE)
Journal of Information and Software Technology http://www.elsevier.com/locate/infsof	Reverse Engineering(WCRE) www.informatik.unitrier.de/~ley/db/conf/wcre/
Empirical Software Engineering Journal http://www.springer.com/computer/swe/journal	International Symposium on Empirical Software Engineering and Measurement(ESEM) http://www.esem-conferences.org/

The second section of Table 3 classifies papers according to their pattern search approach ("Purpose of Study"). Different authors have used these parameters for empirical evaluation of studies in the area of software engineering and reverse engineering. We found only few papers which focused on pure theoretical concepts without any tool support. A large number of papers focus on measuring accuracy of techniques through eliminating *false positives* and by recovering *false negatives* which is the central goal of each presented technique. Most of the papers which compute precision and recall also compare their accuracy with other techniques and claim to have improved precision and recall rates. The last category of papers presented reviews about state of the art in the area of design pattern recovery and highlights strengths and limitations of different techniques.

In the nutshell, pattern recovery approaches are classified into structural analysis, behavioral analysis, semantic analysis and formal specification/composition analysis to recover patterns from the source code of different legacy applications. The papers, in the third section of Table 3 ("Analysis type of study"), are classified on the basis of analysis type used by each design pattern recovery technique. The early approaches used only structural analysis methods to recover patterns and the accuracy of these approaches was very low. We found only a number of papers which recover pattern through pure behavioral analysis. Most of the papers used structural analysis methods supplemented with behavioral analysis to extract patterns with improved accuracy. Semantic analysis methods are used in combination with structural and behavioral analysis to recover patterns which have similar structural and behavioral properties.

We evaluated the scope of studies by the targeted patterns and targeted languages. The forth section of Table 3 ("Scope of Study") gives statistical information about papers based on the type of extracted patterns. The targeted language for most of the papers turned out to be Java with the exception of few papers which examine small case studies of C/C++ systems for the extraction of patterns. The domain of multiple language and language independent design pattern recovery still did not get major attention of researchers.

Lastly, we investigated the state of the art studies on the basis of their accuracy which is the main focus of this paper. The accuracy of pattern recovery techniques is measured using variable parameters as suggested by other authors [9 16]. The papers are classified in the last section of Table 3 ("Accuracy of study") on the basis of precision, recall and F-score. We evaluate accuracy of selected studies on the basis of precision and recall which is discussed in detail in section 5.

Table 3: Statistics and Classification of Literature Review

Section	Technique	No. of References
Object of Study	Design Pattern Recovery	47
	Pattern Recovery and Visualization	4
	Design pattern Formalization	22
Purpose of Study	Theoretical concept	12
	Quantification	32
	Comparison	18
	Review	11
Analysis type of study	Structural analysis	7
	Behavioral analysis	3
	Structural and Behavioral analysis	30
	Structural, Behavioral and Semantic analysis	3
Scope of Study	Creational patterns	3
	Structural patterns	11
	Behavioral patterns	4
	All GOF type patterns	30
Accuracy of Study	Precision	23
	Precision and Recall	15
	No Precision and Recall	9
	F-Score	1

4. Classification and Evaluation Framework

In different studies[16 28 29 30], the frameworks are presented for the evaluation of reverse engineering techniques and tools, but there is still no common framework for evaluating design pattern recovery techniques. Authors in [16] have presented the review of design pattern recovery techniques but they did not focus on evaluation of pattern recovery techniques which is important for empirical evaluation. We adopted a framework for evaluation of design pattern recovery techniques based on key concepts of studies in section 2 and further classification of these studies given in section 3. The underpinning theory of our framework adoption is based on the existing state of the art and from the analysis of similar measures in the broader area of software engineering, reverse engineering and design pattern recovery [29 30 31]. The metrics used in our framework can be extended by researchers and can be used for creating taxonomies. The suggested framework is based on the following dimensions as discussed in the previous section:

1 Object of study
2 Purpose of study
3 Analysis type of study
4 Scope of study
5 Accuracy of study

The stance of this paper is to present the state of the art work at different levels of abstraction that can fulfill requirements of different reviewers. Section 2 classified the state of the art at a higher level of abstraction which can be used for a quick review. Section 3 elaborated the methods used in section 2 in detail based on object, purpose, analysis type, scope and accuracy of each study. We evaluated accuracy of selected approaches in Section 5.

5. Evaluating Accuracy

The accuracy of design pattern recovery techniques is the key concern for the adoptability and reusability of pattern design pattern recovery studies. Different design pattern recovery techniques [1 2 3 4 5] have wide disparity in the results, although they examined the same systems. Petterson et al. [9] presented an approach which evaluates the accuracy of pattern recovery approaches based on different parameters. The authors discuss different factors which influence the accuracy of different pattern recovery approaches but they did not investigate the extracted accuracy of different approaches which is important for the adoption of any methodology. We focus on a micro analysis of *true positives*, *false positives*, and *false negatives* to compare the accuracy of different studies in detail.

The collection of similar examples for different studies was important for the comparison and the evaluation of accuracy extracted by design pattern recovery techniques. We were not able to compare accuracy of all approaches discussed in sections 2&3 because we cannot find the common evaluated examples and recovered patterns for all techniques. We selected seven studies [1 2 3 4 5 6 11] for micro comparison on the basis of the following grounds:

Table: 4 Extracted Pattern Instances

Software	JHotDraw5.1					JUnit3.7					JRefactory2.6.24			Quick UML2001			Apache Ant 1.6.2		
Reference	[2]	[1]	[3]	[4]	[5]	[2]	[1]	[11]	[4]	[5]	[2]	[1]	[6]	[2]	[1]	[3]	[3]	[6]	[2]
Singleton	2	2	x	x	2	0	0	2	x	0	12	2	1	1	1	x	x	1	7
Adapter	18	1	41	4	24	6	0	0	3	6	7	17	16	11	0	27	13	41	4
Composite	1	1	0	0	1	1	1	1	3	1	0	0	x	1	2	0	4	44	14
Decorator	3	1	0	x	3	1	1	1	x	1	1	0	x	0	0	0	0	12	14
Factory Method	3	3	x	x	3	0	0	0	x	0	4	1	0	0	0	x	x	6	38
Observer	5	2	x	x	2	4	3	3	x	2	0	0	x	0	1	x	x	5	0
Prototype	1	2	x	x	1	0	0	0	x	0	0	0	x	7	0	x	x	x	0
Command	0	1	x	x	0	0	2	x	x	0	0	0	0	0	1	x	x	x	x
Template Method	5	2	x	x	5	1	0	0	x	1	17	0	x	5	0	x	x	4	6
Visitor	1	0	x	x	1	0	0	0	x	0	2	2	2	0	0	x	x	1	0
State/Strategy	23	2	x	64	20	3	0	0	6	3	12	2	3	15	0	x	x	26	-
Abstract Factory	x	0	x	x	0	x	0	0	x	0	x	0	x	x	2	x	x	6	x

x: Technique is not used to recover pattern

1 These papers are selected because most of the authors performed experiments on one or more of similar benchmark examples and we can compare the results of different tools on the same examples.
2 Secondly, the source code of these examples is available freely to validate the results of approaches manually or using tools.
3 Thirdly, the size of these systems varies from few lines of source code to over a million lines of code,

which is important to validate the scalability of different approaches.

4 The documentation and executables of tools used by different approaches are available online.

5 Finally, most of these systems have been developed using different design patterns.

4.1 Accuracy Comparisons

The precision and recall metrics are used for the evaluation of information retrieval techniques including design pattern detection approaches. Recall is especially problematic when the number of extracted patterns is large and the false negatives cannot be assured without trusted benchmarks. The relationship between precision and recall metrics determines the correctness of the approaches. Ideally, precision should remain high as recall increases, but in practice this is difficult to achieve [38]. Precision and recall also depend on the type of analysis used by the pattern recovery approaches and is measured on the basis on the basis of the metrics true positives (TP), true negatives (TN),false positives (FP), and false negatives (FN).

Precision and recall are important for measuring the accuracy and the completeness of pattern recovery approaches, but the integration of both factors yield combined effect. Peterson et al. [9] have suggested an integrated common factor for measuring precision and recall metrics for any pattern recovery approach as standard solution to use the weighted harmonic means of P and R(weighted F-Score). They define weighted F-Score Fw, w€ R as:

$$F_w = \frac{(1 + w^2)PR}{w^2P + R}$$

Thus, the highest F-Score is obtained if both precision and recall are high. The suggested value of w =2.28. For precision of 100% and recall of 50%, the value of Fw will be 61% and if precision is 50% and recall is 100% then Fw=72%.

Table 5: Accuracy Comparisions

Reference	Precision (%)	Recall (%)	F-Score (%)
[1]	39	100	80
[2]	100*	100*	100*
[3]	62-97	NM	??
[4]	95	89	90
[5]	94	92	92
[6]	NM	NM	-

NM: Not Mentioned, *: Suspected Values

6. Disparity Analysis

The wide disparity in the results of selected approaches in table 6 motivated us to investigate the causes of disparity in the results of these approaches. This requires the manual analysis of extracted patterns results by comparing complete roles of patterns. Unfortunately, most authors only present information about the number of patterns, but they do not give information about the complete roles and location of these roles in the source code. Even it becomes worse when the extracted roles are not in any standard format because different techniques extract pattern instances using partial or complete roles.

A wide disparity has been noticed in the results of [1 2 3 4 5] on the Adapter pattern in JHotDraw 5.1 as shown in Table 4. These techniques detected 1, 18, 41, 4, 24 instances of the Adaptor pattern respectively which reflect wide disparity in the results of these techniques. It is difficult to realize how many common instances are detected by all approaches because each applied tool displays its results in different formats. Some tools only show the number of patterns extracted and it becomes difficult to realize the extracted location of patterns instances in such tools. We further analysed that major reasons of disparity in adapter patterns are generalization hierarchy and implementation of delegation operation. For example, [2,4, 6] do not take into account all the levels of generalization while detecting patterns from the source code. The level of complexity increases when super classes are implemented as header files. Further, some approaches do not check negative generalization between adaptee class and target class while detecting adapter pattern. Similarly, Dong et. al[16] discussed that developers implement delegation operation using different ways and all approaches do not consider all the possibilities of handling delegation in their pattern detection methods which cause disparity in the results of different approaches. In another example, Table 4 shows the results for the Composite pattern, extracted by [3 6 2] on Apache Ant 1.6.2. The approaches have extracted 4, 14 and 44 instances respectively. It shows that only 4 instances are commonly extracted by all approaches. The dilemma still remains on the realization of common instances. The reasons of different results are handling of aggregation /composition relationship between composite and component classes. Another reason of variation in results of this pattern is its different variants The approaches which detect patterns based on intermediate representation are not able to accurately detect aggregation/compositions between classes which cause disparity in the results of these approaches . Similarly, the approach presented in [4] extracted adapter pattern instances from JHotDraw v6.0b1 with AbstractFigure, NullFigure and HandleEnumeration playing the roles of Target, Adapter and Adaptee classes. The handle method in the NullFigure which delegate request is missing in the AbstractFigure. Moreover, the AbstractFigure is a concrete class which extends to another class. We also tested the standalone version of the DP-Miner tool on Junit 3.8.2 with the XMI file that is available on the website of the tool. It detected “0” results for the adapter and composite

patterns while the author mentions 3 instances of the adapter and composite pattern in their approach presented in [4]. It is important to run each tool individually and analyse its results instead of just comparing results published in papers. Unfortunately, all tools are not available publically for verification of extracted results. Lastly, Table 5 shows the common instances shared by [2 1]. We manually analysed these results by inspecting the source code artifacts which contain none of these common patterns instances. The approaches [2 1] extract 1, 2 instances of the prototype from JHotDraw 5.1 and they are completely different instances. Similar other disparities are also visible from Table 6. The analysis of shared common instances was a very laborious and time consuming task. It is important for the benchmark systems that their published results should be analysed manually before publishing results on their websites. "CI" in Table 6 stands for Common Instances shared by the more than one approach.

Table 6: Disparity Analysis

Software	JHotDraw5.1			JUnit3.7		
Reference	[2]	[1]	[CI]	[2]	[1]	[CI]
Singleton	2	2	2	0	0	0
Adapter	18	1	1	6	0	0
Composite	1	1	1	1	1	1
Decorator	3	1	1	1	1	1
Factory Method	3	3	3	0	0	0
Observer	5	2	1	4	3	2
Prototype	1	2	0	0	0	0
Command	0	1	0	0	2	0
Template Method	5	2	2	1	0	0
Visitor	1	0	0	0	0	0
State/Strategy	23	2	2	3	0	0

6. Critical Analysis and Lessons Learned

Through extensive review of the whole spectrum of research in the area of design patterns and evaluation of the selected techniques we learned the following lessons which are our observations:

1 The standard evaluation frameworks and benchmark systems are very desirable for the evaluation of existing and new design pattern recovery approaches and tools.

2 Most pattern recovery techniques target open source systems for pattern recovery which do not have proper documentation. It is very difficult to compare the results of the techniques because wide disparity exists in the recovered results on same systems. The experiments on commercial and industrial applications are very rarely performed which can realize the application of applied technique.

3 Most techniques experimented only with systems implemented in C++/Java languages to recover patterns. The generalization of these techniques for multiple language pattern recovery is still questionable.

4 Most techniques recover only a few patterns which are relatively easy to detect with good precision and recall, but the real applications are developed using a broader range of patterns. The scalability and generalization of these approaches for complex systems needs to be investigated.

5 The recovered results of design pattern instances are very important for software maintenance and comprehension. The results of pattern recovery techniques reveal the number of recovered patterns, but they do not give any information about the exact location of these patterns in the source code. Furthermore, the visualization of detected design pattern instances is very important for comprehension of examined applications. Pattern recovery approaches paid very little attention on visualization of recovered results.

6 In large applications, parts of the system architecture are based upon patterns, which are interrelated. State of the art pattern detection approaches and tools overlook the detection of composition and overlapping of design patterns.

7 Most approaches performed experiments on one or two examples and they do not cross validate their results against other approaches.

8 Most approaches applied structural and behavioral analysis for pattern recovery with the sole exception of the approaches [4 5]. Through micro analysis of disparity in the results, we came to the conclusion that semantic analysis is very important because a number of patterns have similar structural and behavioral properties and they cannot be correctly recognized without semantic analysis.

9 Little attention is paid on development of common design pattern recovery tools which should support different methodologies. Research community should put efforts to develop tools which should be general and may be integrated with other tools.

10 Last but not least is the requirement of common formalized definitions of all GoF[25] patterns and their variants. The varying definitions hamper the accuracy of pattern recovery approaches.

7. Conclusions

We presented a review on the state of the art techniques used for design pattern recovery with key focus on

evaluating the accuracy of selected techniques. The proposed empirical framework can be used for the evaluation of design pattern recovery techniques and it can be extended by researchers. Our statistical analysis and critical review reveal future research directions as needs in the area of design pattern recovery. Tool developers can use results of this empirical study for developing new design pattern recovery tools. The difficulty of measuring accuracy is due to the unavailability of trusted benchmark systems. Design pattern recovery tools should be developed with standard input/output formats and they should be capable of visualizing extracted pattern information, which is important for the maintenance and comprehension of legacy applications. Furthermore, we realized that researchers should develop new techniques which should be flexible for customization to handle the variations in design pattern detection.

References

[1] Y.-G. Guéhéneuc, and G. Antoniol, "DeMIMA: A Multilayered Approach for Design Pattern Identification", IEEE Transactions on Software Engineering, Vol, 34, No.5, 2008, pp. 667-684.

[2] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design Pattern Detection Using Similarity Scoring", IEEE Transaction on Software Engineering, Vol. 32, No. 11, 2006, pp. 896-909.

[3] A. D. Lucia, V. Deufemia, C. Gravino, and M. Risi, "Design pattern recovery through visual language parsing and source code analysis", Journal of Systems and Software, Vol 82, Issue 7, 2009, pp. 1177–1193.

[4] J. Dong, J. Zhao, and Y. Sun, " A Matrix based Approach to Recovering Design Patterns", IEEE transactions on Systems, Man and Cybernatics, Vol 39, No. 6, 2009, pp. 1271-1282.

[5] G. Rasool, I. Philippow, P. Mader, "Design Pattern Recovery Based on Annotations" International Journal of advances in Engineering Software, Vol 41, Issue 4, 2010, pp. 519-526.

[6] N. Shi, and R. A. Olsoon, "Reverse Engineering of Design Patterns from Java Source Code", In Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06), Vol 00, 2006, pp. 123-134.

[7] Y-G. Gueheneuc, J-Y. Guyomarc'h, and H. Sahraoui, "Improving design-pattern identification: a new approach and an exploratory study", Software Quality Journal, Volume 18, Issue 1, 2010, pp. 145-174.

[8] K. Stencel, and P. Wegrzynowicz, "Detection of Diverse Design Pattern Variants", 15th Asia-Pacific Software Engineering Conference, 2008, pp. 25-32.

[9] N. Pettersson, W. Löwe, and J. Nivre, "Evaluation of Accuracy in Design Pattern Occurrence Detection", IEEE Transactions on Software Engineering, Vol 36, No. 4, 2010, pp. 575-590.

[10] I. Philippow, D. Streitferdt, M. Riebisch, S. Naumann, "An approach for reverse engineering of Design patterns", Journal of Software and System Modeling, Vol 4, No. 1, 2004, pp. 55-70.

[11] O. Kaczor, Y.-G. Gueheneuc, and S. Hamel, "Efficient Identification of Design Patterns with Bit-vector Algorithm", In Proceedings of the Conference on Software Maintenance and Reengineering (CSMR), 2006, pp. 175-184.

[12] D. Beyer and C. Lewerentz, "CrocoPat: efficient pattern analysis in object-oriented programs", In Proceedings of the International Workshop on Program Comprehension (IWPC'03), 2003, pp. 294–295.

[13] J. M. Smith and D. Stotts "SPQR: Flexible automated design pattern extraction from source code", In Proceedings of Automated Software Engineering, 2003, pp. 215-224.

[14] M. Vokac, " An efficient tool for recovering design patterns from C++ code", Journal of Object Technology, Volume 5, No. 1, 2006, pp. 139–157.

[15] H. Lee, H. Youn, and E. Lee, "Automatic Detection of Design Pattern for Reverse Engineering", In Proceedings of 5th ACIS International Conference on Software Engineering Research, Management & Applications, 2007, pp. 577-583.

[16] J. Dong, Y. Zhao, and T. Peng, "A Review of Design Pattern Mining Techniques", International Journal of Software Engineering and Knowledge Engineering (IJSEKE), Vol 16, Issue 6, 2009, pp. 823-855.

[17] A. De Lucia, V. Deufemia, C. Gravino, and M. Risi, "Behavioral Pattern Identification through Visual Language Parsing and Code Instrumentation", In Proceedings of European Conference on Software Maintenance and Reengineering, 2009, pp.99-108.

[18] L. Prechelt, B. Unger, W. F. Tichy, P. Brössler, L. G. Votta, "A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions", IEEE Transactions in Software Engineering, Vol 27, No. 12, 2001, pp. 1134-1144.

[19] N. J. Shafer, W. Wadsack, J.P. Wendehals, and L. Walsh, "Towards pattern design recovery", In Proceedings of International Conference on Software Engineering (ICSE'02),2002, pp. 338–348.

[20] Z. Balanyi, and R. Ferenc, "Mining design patterns from C++ source code", In Proceedings of International Conference on Software Maintenance (ICSM'03), 2003, pp. 305–314.

[21] D. Heuzeroth, T. Holl, G. Hogstrom, and W. Lowe, "Automatic design pattern detection", In Proceedings of the 11th IEEE International Workshop on Program Comprehension, 2003, pp. 94–103.

[22] A. Blewitt, A. Bundy, and I. Stark, "Automatic verification of design patterns in Java", In Proceedings of 20th International Conference on Automated Software Engineering, 2005, pp. 224–232.

[23] R. Keller, R. Shauer, S. Robitaille, and P. Page, "Pattern-based reverse-engineering of design components", In Proceedings of the 21st International Conference on Software Engineering, 1999, pp. 226–235.

[24] J. Paakki, A. Karhinen, J. Gustafsson, L. Nenonen, and A. I. Verkamo, "Software Metrics by Architectural Pattern Mining", In Proceedings of the International Conference on Software: Theory and Practice, 2000, pp. 325–332.

[25] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Massachusetts, 1995.

[26] Ptidej team home page, www.ptidej.net/

[27]http://www.rcost.unisannio.it/wcre2006/colocated_events/DPD4RE.htm

[28] L. J. Fulop, R. Ferenc, and T. Gyimothy "Towards a Benchmark for Evaluating Design Pattern Miner Tools", In Proceedings of the 12th European Conference on Software Maintenance and Reengineering, 2008, pp. 143-152.

[29] Y.-G. Gueheneuc, Y. K. Mens and R. Wuyts, "A Comparative Framework for Design Recovery Tools", In Proceedings of Conference on Software Maintenance and Reengineering (CSMR'06), 2006, pp.123-134.

[30] P. Tonella, M. Torchiano, B. Du Bois and T. Systä, "Empirical studies in reverse engineering: state of the art and future trends", Empirical Software Engineering, , Vol 12, No. 5, 2007 ,pp. 551-571.

[31] D. Sjoberg, J. Hannay, O. Hansen, V. Kampenes A. Karahasanovic, N. Liborg, and A. Rekdal, "A survey of controlled experiments in software engineering", IEEE Transaction in Software Engineering, Vol 3, No. 9, 2005, pp. 733–753.

[32] Nucleus: Nucleus Research Report: Microsoft Patterns and Practices, August 2009 http://msdn.microsoft.com/en-us/practices/ee406167.aspx

[33] C. Wohlin , P. Runeson , M. Höst, M. C. Ohlsson , B. Regnell , A. Wesslén, "Experimentation in software engineering: an introduction", Kluwer Academic Publishers, Norwell, MA, 2000.

[34] W. Wang and V. Tzerpos, "Design pattern detection in Eiffel systems", In Proceedings of 12th Working Conference on Reverse Engineering (WCRE), 2005, pp. 1-10.

[35] D. I.K. Sjoberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal, "A Survey of Controlled Experiments in Software Engineering" ,IEEE Transactions on Software Engineering, Vol. 31, No. 9, 2005, pp. 733-753.

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

[37] H. Huang, S. Zhang, J. Cao, and Y. Duan, "A practical pattern recovery approach based on both structural and behavioral analysis", Journal of Systems and Software, 75(1-2), 2005, pp.69-87.

[38] C. K. Roy, "Detection and Analysis of Near-miss Software Clones", PhD thesis, pp. 164, Queen's University Kingston, Ontario, Canada, August 2009.

[39] C. Park, Y. Kang, C. Wu, and K. Yi, "A static reference analysis to understand design pattern behavior", In Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04), 2004.

[40] P. Tonella, G. Antoniol, "Object oriented design pattern inference." In Proceedings of International Conference on Software Maintenance (ICSM'99), 1999.

[41] F. Arcelli and L. Cristina, "Enhancing Software Evolution through Design Pattern Detection", In Proceedings of the 3rd International Workshop on Software Evolvability (PCODA'08), 2007, pp. 7-14.

[42] C. K. Roy, "Detection and Analysis of Near-miss Software Clones", PhD thesis, pp. 164, Queen's University Kingston, Ontario, Canada, August 2009.

[43] Fujaba Home Page :< http://wwwcs.unipaderborn.de/cs/fujaba

[44] Scientific Toolworks Inc. Understand for C++, 2003, http://www.scitools.com/.

[45] I. Bayley and H. Zhu , "On the Composition of Design Patterns", In Proceedings of Eighth International Conference on Quality Software, 2008, pp. 27-36 .

[46] R. B. France, D.-K. Kim, S. Ghosh and E. Song, "A UML-Based Pattern Specification Technique," IEEE Transactions on Software Engineering, Volume 30, No. 3, 2004, pp. 193-206.

[47] Y. Wang, and J. Huang, "Formal Modeling and Specification of Design Patterns Using RtPA", International Journal of Cognitive Informatics and Natural Intelligence, Volume 2, Issue 1, 2008, pp. 100-111.

[48] T. Tabi, D. Chek and L. Ng, "Modeling of Distributed Objects Computing Design Pattern Combinations using Formal Specification Language", International Journal of Applied Mathematics and Computer Science, Volume 13, No. 2, 2003, pp. 239-253.

[49] G. Kniesel, and A. Binun," Standing on the Shoulders of Giants -A Data Fusion Approach to Design Pattern Detection", In Proceedings of 17th International Conference on Program Comprehension, 2009, pp. 208-217.

[50] G. Antoniol, R. Fiutem, and L. Cristoforetti, , "Design pattern recovery in object-oriented software", In Proceedings of the 6th international workshop on program comprehension , 1998, pp. 153–160.

[51] M. V. Detten, and S. Becker, "Combining Clustering and Pattern Detection for the Reengineering of Component-based Software Systems", In Proceedings of the 7th International Conference on the Quality of Software Architectures, QoSA ,pp. 23-32, 2011.

Ghulam Rasool did his PhD from TU Ilmenau, Germany in March 2011. Dr. Rasool is recently working as Assistant Professor at the Department of Computer Science, Comsats Institute of Information Technology, Lahore Campus. The topics of his research are reverse engineering, design patterns recovery, program comprehension and source code analysis. He is actively engaged in different research projects and has published number of papers in different conferences and journals. He also is reviewers of different international journals.

Detlef Streitferdt did his PhD from T|U Ilmenau in 2004. He is recently working as Professor at Software Systems/Process Informatics group in TU Ilmenau, Germany. He has vast experience of working in different industries and has actively participated in different collaborative research projects. His area of research includes Software architectures, Software Products Lines, Requirement Engineering and Goal Oriented Requirement Engineering. He organized number of conferences and workshops.