Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 3 (2011) 813–819

Procedia Computer Science

www.elsevier.com/locate/procedia

WCIT-2010

Evaluation of design pattern recovery tools

Ghulam Rasool[a, b] *, Patrick Maeder[a], Ilka Philippow[a]

[a]TU Ilmenau, Software Systems/Process Informatic,Ilmenau ,Germany
[b]Comsats Institute of Information Technology, Department of Computer Science, Lahore, Pakistan

Abstract

Design pattern recovery approaches are assisted by different tools which recognize patterns from source code of legacy applications. Several tools are presented in related work, but little attention is paid on the evaluation of tools due to the unavailability of standard benchmarks and frameworks. Different pattern recovery tools extract different results from the same examined systems. The causes for disparity of results and ignorance for cross validation of results by different tools is still not thoroughly investigated. In this paper, we review existing design pattern recovery tools based on their available features and compare them regarding limitations of different tools. We recommend guidelines based on our observation and on the evaluation of different tools which can be used for comparing features of existing tools and developing new design pattern recovery tools.

Selection and/or peer-review under responsibility of the Guest Editor.

Keywords:Design Patterns; Pattern recovery tools; Tool analysis; Program comprehension; Reverse Engineering

1. Introduction

The application of design pattern recovery approaches and tools is widely recognized by the research community because they supplement program comprehension, maintenance, refactoring, restructuring, reverse engineering and re-engineering legacy applications. The overall worth of design pattern recovery approaches is measured by the precision and recall of extracted pattern instances. Design pattern recovery tools are used for extracting pattern instances from source code of legacy applications. Many tools [1 2 3 4 5 6 7 8 9 10] are used for the recognition of patterns in past staring from last two decades, but still the reliability of results is questionable for selection and application of tools. The evaluation and selection of tools has become an important concern for all stakeholders including companies, customers and users. The results of evaluation can be used for the iterative development of tools. Most of the tools differ in precision and recall, in input and output formats, in abstraction and extraction techniques and in some other features. The foremost question to answer is "What is the definition of the recognized design patterns used by different tools during pattern recovery process"? The abundance of tools needs their evaluation to explore the factors that hamper accuracy of tools. Most serious concerns are different results of different tools on the same examined systems.

The code level and model level support have been noticed in different tools used for pattern recognition. The code level tools directly take the source code as input and use different techniques (AST, ASG, RSF, PDG etc.) to represent the source code. These tools use different algorithms to extract patterns from the representation of the

* Ghulam Rasool. Tel.: +49-3677-69-1252; fax: +49-3677-69-1220.
E-mail address: ghulam.rasool@tu-ilmenau.de.

1877-0509
doi:10.1016/j.procs.2010.12.134

source code. These tools report more errors because the hard coded algorithms used by these tools are not able to detect the implementation variants of design patterns. The model level tools such as IBM Rational Rose[11], Enterprise Architect Modeling tool[12] etc. are used to create the intermediate representation of the source code in the form of different models and then use different techniques to extract patterns from data available in models. The accuracy of these tools depends on the ability of the modeling tool. To the best of our knowledge, modeling tools are still not able to extract all the relationships from source code. Both levels of tools have their strengths and limitations. The integrated use of code level and model level support is ideal to reduce errors in pattern detection.

Commercial, academic and open source tools exist in literature but the important question is to get accurate information about these tools. It is necessary to collect, organize and analyze several sources of information related with pattern recovery tools for their evaluation. Design pattern tools have different features for evaluation like extraction, abstraction, presentation, performance, scalability and accuracy etc., but we focus our analysis on evaluating the precision and recall of different tools. Some tools perform experiments on very small toy examples using few patterns and claim very high precision and recall rates but simply using easy examples silos misses part of picture because; they lack support of solid concept and scalability.

It is important to evaluate existing tools before developing new tools in order to avoid reinvention of similar tools which are already available in the market. Most of the research prototyping tools are developed to implement specific methodologies and they can not be integrated with other tools due to different input/output formats. The application of such tools only serves the benefits to publication of single methodology and do not give any benefit to the community in future. It is obvious from analysis of literature that little attention is paid on developing tools which should be integrated with other tools. Finally, little attention is paid on developing open source tools.

2. Related Work

An Analysis of existing pattern recovery tools has been performed in previous studies with different objectives, such as the ones detailed below:

Fulop et al. [13] have presented an evaluation and comparison of the design pattern recovery tools Columbus, Maisa and CrocoPat on four C++ open source systems (DC++, WinMerge, Jikes and Mozilla). The comparison focused on speed, memory consumption and the difference between the hits of used tools. The precision and recall of tools are not discussed in the evaluation. There is also no cross validation of results which is important to explore disparity of results. The same authors have presented a benchmark for evaluating and comparing design pattern recovery tools. They evaluated three tools: Columbus (C++), Maisa (C++), and Design Pattern detection Tool (Java) on open source softwares (Mozilla, NotePad++, JHotDraw, JRefactory and JUnit). The focus of authors was to build a large database of design pattern results extracted from different open source systems. The community effort is required to realize the goal of authors. They uploaded 1172 design pattern instances with true positives and false positives. The authors compare the average precision and recall of each tool on all the examined systems.

Gueheneuc et al. [14] introduced a comparative framework for design recovery tools. The generic framework can be used for evaluating design pattern recovery tools and is quite helpful for developing new design pattern recovery tools. The framework is based on eight factors: context, intent, users, input, technique, output, implementation and tool. Each factor has further different parameters. The authors compare their self developed tool Ptidej [4] with LiCoR (Library for Code Reasoning [16]) on the basis of eight mentioned factors. We focus on evaluating the recent state of the art tools on the basis of their precision and recall.

Petterson et al. [15] have presented an approach to evaluate the accuracy of pattern recovery tools. The authors discuss different problems like pattern variants, pattern instance type, exact and partial matches, the system size, precision and recall, and control sets which affect the accuracy of pattern recovery tools. The approach focuses on a controlled set to be used as benchmark for evaluating the results of different tools. The initial benchmark contains results for a few patterns on different examples (SWT 3.1.0, JHotDraw 5.1).

Wang et al. [9] presented a design pattern recovery tool DPVK and compared it with other tools (Pat, KT, SPOOL, and BOORET). The evaluation focuses on the supported language, analysis approach, by products, and effectiveness/efficiency attributes of examined tools. The tools are not evaluated on real examples and precision/recall is not considered in the evaluation.

A comparison of reverse engineering tools based on design pattern decomposition is presented in [17]. The authors compare Fujaba and SPQR regarding how sub-patterns can improve the detection capability of these tools

used for design pattern detection. The composite pattern is presented as an example. Authors discuss general characteristics of both tools including their extensibility, formalization aspects and variant handling etc., but they do not discuss the evaluation of tools based on their extracted pattern instances.

3. Selection of Tools

We reviewed 30 tools used for design pattern recovery in literature and selected six tools for comparison and evaluation on the basis of available information such as (documentation, executable source code, extracted pattern instances, precision, recall, published in well known conference/journal etc.). We filtered out a number of tools which are developed and applied before 2003, because we think that new tools are developed on the base of modern technologies. Similarly, we filtered tools which are examined on very small examples and have extracted only few patterns which are relatively easy to detect. The statics of selected tools are given in table 1. A brief description about tools is presented below:

3.1. DPRE(Design Pattern Recovery Environment)

DPRE is a research prototype developed at the University of Salerno, Italy. It is generated from a grammar specification by using the visual language grammar VLDesk (Visual Language Desk) system. The main components of the VLDesk architecture are: the Symbol Editor, the Visual Production Editor, the Textual Production Editor and the Visual Programming Environment Generator. DPRE is a general purpose tool which supports other reengineering activities. It includes a UML class diagram editor and visualizes the imported class diagrams. DPRE is implemented as Eclipse Plug-In. It suffers the problem of scalability and the disparity of results is in recovered patterns is noticed.

3.2. DeMIMA (Design Motif Identification Multilayered Approach)

DeMIMA is a prototyping tool developed at the University of Montreal. The tool is implemented on top of the PTIDEJ framework using Java programming language. It takes benefits from existing libraries of PTIDEJ [4] to implement the constraint resolver. It detects design motifs by highlighting microarchitectures which are helpful for the comprehension and other reengineering activities. Explanation-based constraints programming is used to identify microarchitecutres for the maintainers in order to direct their search and discriminate between the false positives. The tool is tested on 33 industrial components as well as open source systems. DeMIMA is extensible, scalable and ensures traceability between the implementation and design artifacts.

3.3. DPD :(Design Pattern Detection v4.3)

DPD is a research prototype developed at the University of Macedonia. It is implemented in Java and employs the java byte code manipulation framework, which extracts the static structure of the legacy system including abstraction, inheritance, constructor signatures, method signatures, method invocations and object instantiation. This primary information is used to extract more advanced properties of design patterns like collection element type checking, similar abstract method invocation, abstract method adaption and static self reference etc. The extracted information is used to generate the matrix that describes the system to be examined. In the current implementation, pattern descriptions are hard-coded within the program. The tool extracts fast results on medium size examples but it hang sup on large software systems.

3.4. PTIDEJ :(Pattern Traces Identification, Detection, and Enhancement in Java)

The PTIDEJ system is an automated open source system implemented in Java which uses approximate and exact matches to extract different idioms, micro-architectures, design patterns and design defects from source code. It has been initiated at the University of Montreal. The layered architecture used in the implementation of PTIDEJ provides flexibility for its extension. It uses the PADL meta-model (Pattern and Abstract level Description Language) for describing models of a program. It has parsers for representing meta models of AOL, C++ and Java.

It is a tool suite which is generic and used for the analysis and maintenance of object-oriented architectures.

3.5. PINOT :(Pattern Inference and Recovery Tool)

PINOT is developed at the University of California as a research prototype. It is fully automated design pattern recovery tool which is implemented as modification of Jike (an open source java compiler written in C++). It is a command line tool. PINOT detects all the GOF [18] patterns except the prototype, builder and iterator. PINOT reduces the search space for detecting patterns by selecting the properties of patterns playing key role in an order. It does not recognize any user defined or user-extended data structures. The tool is not customizable because it uses algorithms which are hard coded in the source code.

3.6. DP-Miner:(Design Pattern Miner)

DP-Miner has been developed at the University of Taxas, Dallas as a research prototype. It is intended to recover design patterns applied in software from source code. DP-Miner analyzes structural, behavioral, and semantic characteristics of system and patterns. In structural analysis, DP-Miner presents system structure in a matrix with the columns and rows to be all classes of the system. The value of each cell represents the relationships among classes. The structure of each design pattern is similarly represented in another matrix. The discovery of design patterns from source code becomes possible by matching the two matrices.

Table 1. Statistics and features of different tools

Tool	Doc	ESC	Website	Case Studies	Patterns Detected	P&R (%)	Publication
DPRE	Yes	Yes	http://www.sesa.dmi.unisa.it/dpr	JHotdraw5.1,Apache Ant JHotdraw6.0b1 etc.	AD, PR, BR, DE, CM, FA, FL	65-97/NM	[1]
DeMIMA	Yes	No	(http://www.ptidej.net/research/demima/	JHotDraw 5.1, JUnit3.7 Quick UML2001 etc.	SI, FM, AD, DE, CP, CD,VR, OB,TM, ST/ SR	34/100	[3]
DPD	No	Yes	http://java.uom.gr/~nikos/pattern-detection.html	JHotDraw 5.1,JUnit3.7, JRefactory etc.	SI, FM, AD, DE, CP, CD,VR, OB,TM, ST/ SR	95/100	[2]
PTIDEJ	Yes	Yes	http://ptidej.iro.umontreal.ca/	JHotDraw 5.1 ,QuickUML 2001etc.	SI, FM, AD, DE, CP, CD,VR, OB,TM, ST/ SR	NM/75	[4]
PINOT	No	Yes	http://www.cs.ucdavis.edu/~shini/research/pinot/	Apache Ant 1.6.2, JHotDraw6.0b1, AWT etc.	GOF patterns except(PT, BU,IT,)	NM/NM	[5]
DP-Miner	Yes	Yes	http://www.utdallas.edu/~yxz045100/DesignPattern/DP_Miner/index.html	JAWT, JEdit, JHotDraw etc.	AD, BR, CP, ST, SR	95/97	[6]

PT: Prototype **SI:** Singleton **FM:** Factory Method **BU:** Builder **AD:** Adapter **CP:** Composite **PX:** Proxy **DC:** Decorator **BR:** Bridge **FA:** Façade **FL:** Flyweight **VR:** Visitor **TM:** Template Method **CD:** Command **OB:** Observer **ST:** State **SR:** Strategy **IT:** Iterator **NM:** Not Mentioned Doc: Documentation **ESC**: Executable Source Code **P&R**: Precision and Recall

4. Evaluation of Tools

The collection of similar examples for the different tools was important for the comparison and the evaluation. The “x” in table 2 shows that an approach is not able to detect pattern. We selected systems for experiments as mentioned in table 2 due to following reasons:

- These systems are selected because most of selected tools performed experiments on one or more of these systems and we can compare results of different tools on the same systems.
- Secondly, the source code of these applications is available freely to perform experiments.
- 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 tools.

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

Table 2. Comparison of results by different tools

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

5. Discussion on Results

A wide disparity has been noticed in the results of [1 2 3] on the Adapter Pattern in JHotDraw 5.1. The tools detected 41, 18, 1 instances of Adaptor which show the wide disparity in the results of these tools. There is only one common instance detected by all approaches and it is also difficult to realize that common instance because each tool displays its results in different formats. Some tools only show the number of pattern extracted and it becomes difficult to understand the extract location of patterns in such tools. Similarly, the approach and tool presented in [1] extract adapter pattern from JHotDraw v6.0b1 with AbstractFigure, NullFigure and HandleEnumeration playing roles of Target, Adapter and Adaptee classes. The handle method in the NullFigure which delegate request is missing in the AbstracFigure. Moreover, the AbstractFigure is a concrete class which extends to another class. We tested the stand alone version of DP-Miner tool on JUnit3.8.2 with the XMI file that is available on the website of tool. The tool detects “0” results for the adapter and composite patterns while the author mentions 3 instances of adapter and composite in their approach presented in [6].

6. Critical Review and Observations

- Most of the tools target open source systems for pattern recovery which do not have proper documentation. It is very difficult to compare results of the tools because wide disparity exists in the recovered results.
- Most of tools perform experiments only on systems implemented in C++/Java languages for pattern recovery.
- Some tools recover only a few patterns with good precision and recall, but the real applications are developed using broader range of patterns.
- The recovered pattern results are very important for program comprehension. The numbers of pattern recovery tools only give results about the number of recovered patterns but do not give any information about the exact location of these patterns in the source code.
- In real applications, smaller patterns are composed to build the larger patterns. State of art pattern detection approaches and tools overlook the detection of composition and overlapping of design patterns.

7. Recommendations

- The benchmark systems are very desirable for evaluating and cross validating results of different tools.

- The tools should be developed with standard input/output formats for integration with other tools.
- The visualization support is important for the comprehension of extracted pattern instances.
- Tools should be flexible for customization to detect the implementation variants of design patterns.
- The tools should be able to detect patterns from source code of multiple languages.
- The tools should be tested on open source as well as industrial applications for measuring precision and recall.

8. Conclusions

We presented an extensive study on the recent six tools used for design pattern recovery with major focus on evaluating the precision and recall of the examined tools. We reviewed the features of selected tools and presented our observations and recommendations which can be used for comparing existing tools and developing new tools. The difficulty of measuring precision and recall is due to unavailability of trusted benchmark systems.

References

1. Lucia, A.D., Deufemia, V., Gravino, C., and Risi, M., Design pattern recovery through visual language parsing and source code analysis, The Journal of Systems and Software, Vol 82, pp. 1177–1193, 2009.
2. Tsantalis, N., Chatzigeorgiou, A., Stephanides, G., and Halkidis, S.T., Design Pattern Detection Using Similarity Scoring, IEEE Transaction on Software Engineering, Vol. 32, No. 11, pp. 896-909, November 2006.
3. Guéhéneuc, Y-G., and Antoniol, G., DeMIMA: A Multilayered Approach for Design Pattern Identification, IEEE Transactions on Software Engineering, Vol, 34 No.5, pp. 667-684, 2008.
4. Gueheneuc, Y-G., Sahraoui, H., and Zaidi, F., Fingerprinting design patterns, In Proceedings of the 11th Working Conference on Reverse Engineering (WCRE), pp. 172-181, Victoria, BC. Canada ,November, 2004.
5. Shi, N., and Olsoon, R.A., Reverse Engineering of Design Patterns from Java Source Code, In Procedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06), Vol 00, pp. 123-134, 2006.
6. Dong, J., Lad, D.S., and Zhao, Y., DP-Miner: Design Pattern Discovery Using Matrix, In Proceedings of the Fourteenth Annual IEEE International Conference on Engineering of Computer Based Systems (ECBS), pp. 371-380, Arizona, USA, March 2007.
7. Smith, J.M., and Stotts, D., Elemental Design Patterns: Case Studies in Automated Design Pattern Detection in C++ Code using SPQR, Technical Report TR05-013, Univ. of North Carolina, 2004.
8. Beyer, D., Lewerentz, C., CrocoPat: efficient pattern analysis in object-oriented programs. In: Proceedings of the International Workshop on Program Comprehension (IWPC'03), Portland, OR, USA, pp. 294–295.
9. Wang, W., and Tzerpos, V., DPVK - An Eclipse Plug-in to Detect Design Patterns in Eiffel Systems, Electronic Notes in Theoretical Computer Science (ENTCS), Volume 107, pp. 71-86, 2004.
10. Balanyi, Z., Ferenc, R., Mining design patterns from C++ source code. In Proceedings of International Conference on Software Maintenance (ICSM'03), Amsterdam, The Netherlands, pp. 305–314, 2003.
11. Sparx System Architect Modeling tool. <http://www.sparxsystems.com/products/ea/> [accessed 05.05.09].
12. IBM Rational Rose Website, http://www.ibm.com/software/rational/ [accessed 10.06.10].
13. Fulop, L.J., Ferenc, R., and Gyimothy, T., Towards a Benchmark for Evaluating Design Pattern Miner Tools, In Proceedings of the 12th European Conference on Software Maintenance and Reengineering, pp. 143-152, Athens, Greece, 1-4 April, 2008,
14. Gueheneuc, Y., Mens, K., and Wuyts, R., A Comparative Framework for Design Recovery Tools , In Proceedings of Conference on Software Maintenance and Reengineering (CSMR'06), pp.123-134, Bari Italy, March 22-24, 2006
15. Pettersson, N., Löwe, W., and Nivre, J., On Evaluation of Accuracy in Pattern Detection, First International Workshop on Design Pattern Detection for Reverse Engineering (DPD4RE'06), pp. 1-8, October 2006.
16. The Licor Homepage, http://prog.vub.ac.be/research/DMP/soul/soul2.html.
17. Arcelli, F., Masiero, S., Raibulet, C., and Tisato, F., A Comparison of Reverse Engineering Tools based on Design Pattern Decomposition, In Proceeding of Australian Software Engineering Conference (ASWEC'05), pp.262-269, Brisbane, Australia, March 29-April 01, 2005.

18. Gamma, E., Helm, R., Johnson, R., and Vlissides, J.M., “Design Patterns: Elements of Reusable Object Oriented Software”, Addison-Wesley Publishing Company, Reading, MA, 1995.