Design Patterns: Current Challenges, Trends, and Research Directions

Alireza Rouhi
Bahman Zamani
Abstract: Design patterns solve recurring design problems. We can classify the literature on design patterns into two main categories: 1) Pattern Application, i.e., applying design patterns on a design model which is a forward engineering approach, and 2) Pattern Detection and Identification, i.e., extracting applied design patterns from a given source code which is a reverse engineering approach. With the above classification in mind, this report intends to explore the current challenges behind the design patterns and related tools.
Abstract: Design patterns solve recurring design problems. We can classify the literature on design patterns into two main categories: 1) Pattern Application, i.e., applying design patterns on a design model which is a forward engineering approach, and 2) Pattern Detection and Identification, i.e, extracting applied design patterns from a given source code which is a reverse engineering approach. With the above classification in mind, this report intends to explore the current challenges behind the design patterns and related tools.

1 Introduction

Patterns are solutions to recurring design problems [GHJV94, SFJ96]. In this report, to investigate the issues regarding design patterns, we move on to the current state of design patterns’ challenges, trends, and research directions. Based on studying the past and the current researches, we classify the design pattern research field into two broad categories: 1) design pattern application 2) design pattern detection and identification (see Figure 1). Design Pattern Application is aimed at applying pattern solutions in the software design, i.e., it is a forward engineering approach. However, the Pattern Detection and Identification is a backward and reverse engineering approach which intends to detect and identify the applied design patterns in the given source codes. As we know, a design pattern as a solution to a recurring problem has several components and constituent elements such as the pattern intent, motivation, applicability/known uses, and consequences, to name a few [GHJV94]. Thus, most of the existing pattern detection tools use other terms such as design motifs in [Gué07a], i.e. the solution components of design patterns which can be extracted from the source code or design model in a more tangible way.

In the following, we will explore the details of these techniques through investigating their practical applications and related researches and studies.
2 Pattern Application

The main drawbacks on the way of pattern application include: expertise and knowledge requirements in the application of correct design patterns; lack of formal basis and techniques to aid in the systematic pattern selection and application; and little support to verify the correctness of the applied design patterns and their relationships [Kim08, FBB+14].

2.1 Transformation-based Pattern Application

In order to apply systematically a design pattern to a model, [Lan14] uses model transformation for the application and verification of design patterns in a design model. Here, first a model is developed regardless of any design pattern in mind and then in the next step, the produced model can be refactored to improve its quality through the introduction of suitable design patterns. Finally, model transformation verification can be used to verify that the existing model restrictions are preserved.

In order to provide qualitative and continuous services in the cloud-based systems, [AB14] applies a self-adaptive architecture based on the adaptation patterns in the form of a Pattern Language (PL) recovered from the architecture adaptation log of the underlie running system.

Refactoring a design model using design patterns has been experienced in [FCSK03]. Here, based on the defined metamodel of the design pattern with matching its problem specification against the design model, refactoring automated through transformation on the level of the design pattern metamodel.
2.2 Applying PLs in Practice

With the emergence of patterns, pattern collections and PLs in various contexts, designers are encouraged to use the PLs in a practical sense to solve common software problems as a whole [HGY14].

2.2.1 Providing a framework for applying patterns of PLs in software development

Applying suitable and correct patterns of several and inter-connected PLs in practice together with solving a real business and practical case study have been presented in [HGY14]. The authors in [HGY14] claims that usage of their proposed framework for applying patterns not only will provide solutions to specific design problems but also will help them on how, when, and on what order to apply patterns. The main steps of [HGY14] framework are (1) adapting existing patterns first, if it is possible and there are matched patterns with the constraints of current problem context, and (2) if there is not any pattern match, divide the problem to subproblems and repeat the step (1) again.

2.2.2 Pattern Language Verifiers (PLVs) in Model-Driven Design

Use of a unique “Sign” stereotype as the indication of a single applied pattern which is specified by a model designer, [Zam09] verifies the used pattern structure and its relationships with other applied patterns through the proposed PLV modules, i.e., lexical, syntactical and semantical verification modules. In other words, a design model is developed using patterns of a PL, i.e. Patterns of Enterprise Application Architecture (PoEAA), then the PLV, named ArgoPLV, verifies that whether the applied patterns and their relationships are correct or not.

3 Pattern Detection and Identification

Exploring the researches which have been conducted in the last two decades reveals many projects which aimed at the detection and identification of the design patterns in the existing design model or source code [AFC98, BP00, AACGJ01, AAG01, GJ01, Don02, HHHL03, CDLD+05, TCSH06, Gue07a, GA08, SW08, vDMT10b, AFZ11, AFMR11, FZM11, EBM12, EBL13, FMR13, AZY14, YZC15, ZFS15]. This is typical because of the philosophy of software patterns which is to detect and identify the best practices from the existing models or codes to convey them as recurring solutions.
to the same problems in the same domain and problem space in the future. Of course, this process is an approximate approach. Therefore, probably the detected and identified pattern instances will be approximate and hence controversial and debatable. Usually, two metrics are used to evaluate the pattern detection tools and methods applicability: precision and recall. These metrics are defined in [Gué07a] as follows (Note: here, a design motif is formed from an inter-class structure which per se is constructed with some defined micro-architectures/µA):

\[
\text{precision} = \frac{|\{\text{true } \mu A\} \cap \{\text{identified } \mu A\}|}{|\{\text{identified } \mu A\}|} \quad (1)
\]

\[
\text{recall} = \frac{|\{\text{true } \mu A\} \cap \{\text{identified } \mu A\}|}{|\{\text{true } \mu A\}|} \quad (2)
\]

Design pattern detection and identification is a reverse engineering approach which helps to analyze and understand the legacy code design and intentions for maintenance and upgrading purposes as well as transferring and reusing in between design knowledge in the development of similar projects [vDMT10b, AZY14, YZC15]. Therefore, many tools regarding pattern detection and identification have been presented to help designers understand and analyze the details of the existing source code.

Authors in [AFC98] present a design pattern recovery process which tries to recover five of seven structural design patterns of the Gang-of-Four (GoF) [GHJV94] book, i.e. the adapter, the proxy, the composite, the bridge and the decorator in the object-oriented software development and design. Here, a four step process has been introduced which extracts design patterns using the structural properties of the corresponding class diagrams. The mentioned process is depicted in Figure 2. To cope with the combinatorial state explosion problem of the involved classes and in between relationships of the detected patterns, a multi-stage reduction technique with the exploitation of the metrics and constraints of classes and their relations is used to reduce the matched candidates in the final result of extracted design patterns. Due to the conservative aspect which is taken by the proposed process, it is impossible to have the false negative results in the detected patterns. However, there maybe some false positive results in the recovered design patterns which needs manual inspection to verify from the expert and designer side.

The basic design pattern detection techniques are shown in the Figure 3. According to the literature, the pattern detection analysis approach taken by most of the tools is static or dynamic. Furthermore, static or dynamic analysis is accomplished structural, considering only the structural elements of the programs or behavioral which takes into account the program execution traces too. Due to the important
The role of behavioral elements in the design pattern specification, the false positive rate of pattern detection results can be reduced with the combination of both static and dynamic analysis as well as structural and behavioral analysis [Bin12].

Due to the structural similarity between patterns like the Strategy and State in the GoF catalog [GHJV94], using only static analysis can increase the pattern detection false positive rate. As a result, the analysis approach must take into account both of the static and dynamic analysis in order to distinguish correctly aforementioned similar patterns. Following this section, we introduce some of the important and invaluable tools in the pattern detection field.

<table>
<thead>
<tr>
<th>What to Analyze</th>
<th>Structure</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Analysis (program)</td>
<td>types</td>
<td>call graph</td>
</tr>
<tr>
<td></td>
<td>fields and method signatures</td>
<td>data flow</td>
</tr>
<tr>
<td></td>
<td>associations, inheritance</td>
<td></td>
</tr>
<tr>
<td>Dynamic Analysis (traces)</td>
<td>object relationships</td>
<td>execution traces</td>
</tr>
<tr>
<td></td>
<td></td>
<td>object creation</td>
</tr>
</tbody>
</table>

Figure 3: The basic design pattern detection techniques (adapted from [KB09])
3.1 PTIDEJ

The Pattern Trace Identification, Detection, and Enhancement in Java (PTIDEJ)\(^1\) is also a reverse engineering tool suite which contains several modules aimed at the identification of design patterns applied in multiple source code formats specifically programs written in Java [Gué07b].

3.2 Reclipse

The Reclipse \(^2\), based on the From UML to Java And Back Again (FUJABA), is a reverse engineering tool suite aimed at detecting the GoF design patterns in the Java source code has been developed as an eclipse plug-in at the University of Paderborn. This tool integrates structural analysis with a subsequent dynamic analysis to recover design patterns which are similar in their structure too, such as the State and Strategy design patterns. Figure 4 illustrates the pattern detection steps involved in the Reclipse tool suite. Here, to have a formal and independent notation for the specification of design patterns, a metamodel has been developed [vDMT10b, vDMT10a].

3.3 DPJF

Recently, a novel research with a relatively complete literature on the design pattern detection tools has been done by Alexander Binun at the University of Bonn [Bin12]. One of the contribution of this research is presenting a tool named Detection of Patterns by Joining Forces (DPJF)\(^3\) which is claimed that it has won the game of precision and recall in competing with the contemporary tools in the design pattern detection of various Java coded benchmarks (see Figure 5). Furthermore, DPJF has a

\(^1\)http://www.ptidej.net/tools/
\(^2\)http://www.fujaba.de/projects/reengineering/reclipse.html
\(^3\)https://sewiki.iai.uni-bonn.de/research/dpd/dpjf/start
reasonable speed too [BK12]. To obtain qualitative pattern detection results in terms of precision and recall as well as improving the query speed, [Bin12] applies both of structural and behavioral analyses to detect design motifs in the source code.

A summary of the literature and state of research on the design pattern detection and related tools are illustrated in Figure 6.

### 3.4 VPML

Visual Pattern Modeling Language (VPML) [EBL13] intends to detect design patterns on the Meta-Object Facility (MOF)\(^4\)-based modeling languages. Each pattern is modeled using VPML. Then, the modeled pattern through mapping to a Query, View, Transformations (QVT)-R\(^5\) transformation, runs on a given model to detect its used pattern instances. [EBL13] claims that its presented approach with prototyping on the Eclipse detects some of the GoF design patterns [GHJV94] on the given Unified Modeling Language (UML) models and Control Flow patterns on the given Business Process Model Notation (BPMN)\(^6\) models.

---

\(^4\)http://www.omg.org/spec/MOF/

\(^5\)http://www.omg.org/spec/QVT/

\(^6\)http://www.omg.org/spec/BPMN/
3.5 MARPLE-DPD

Metrics and ARchitecture Reconstruction PLugin for Eclipse-Design Pattern Detection (MARPLE-DPD) is a tool suite which detects design patterns in the Java source codes by using machine learning techniques (see Figure 7 for more details on its architecture). After analyzing the source code and constructing the Abstract Syntax Tree (AST), the detection is performed by the other two steps named Joiner (which collects detected pattern instances) and Classifier (which determines the correct and incorrect detected candidate patterns).

To the best of our knowledge, none of the aforementioned tools [Gué07b, vDMT10b, Bin12] consider the relationships between detected design patterns.

4 Conclusion

In this report, we classified the state of research on design patterns into two main categories: 1) pattern application, and 2) pattern detection and identification. We tried to explore and present current challenges and issues regarding the aforemen-
tioned classification. Although, there are several well known support tools to detect and identify applied patterns in the source code (like Eclipse\cite{vDMT10b}, PTIDEJ\cite{Gue07b}, DPJF\cite{Bin12}, and MARPLE-DPD\cite{ZFS15}, to name a few), lack of a unified and popular formalism is one of the main obstacles on applying design patterns in general\cite{Zdu07,Zam09,ZB13}. Note that most of the current commonly used design patterns have been presented in natural languages. Also, to the best of our knowledge, there is not any considerable research and supporting tools regarding pattern inter-relationships in general and PLs in particular.

As a result, presenting a unified formal notation to representing design patterns and pattern inter-relationships will be one of the main research challenges in the near future. With a formal notation in hand for patterns and pattern inter-relationships, it will be easy to develop support tools in this field.

References


[AB14] Aakash Ahmad and Muhammad Ali Babar. Towards a pattern language


[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, 1994. 1, 4, 5, 7


