A Metric for Measuring the Abstraction Level of Design Patterns

Atsuto Kubo  
Waseda University  
Japan  
a.kubo@fuka.info  
waseda.ac.jp

Hironori Washizaki  
Waseda University  
Japan  
washizaki@waseda.jp

Yoshiaki Fukazawa  
Waseda University  
Japan  
fukazawa@waseda.jp

ABSTRACT

The abstraction level of the problem treated by a design pattern has wide variety, from architecture to near implementation. There is no objective metric indicating the abstraction level of the problems addressed by patterns. Thus, it is difficult to understand the abstraction level of each pattern and to position a new pattern. In this paper, a metric is proposed. It indicates the relative abstraction level of a pattern’s problem. We propose a metric obtained from inter-pattern relationships. We also propose a visualization method for the metric. Using such metrics, we aim to help developers easily understand the abstraction level of each pattern and, therefore, to better decide about its usefulness for the problem at hand.

Categories and Subject Descriptors

D.2.11 [Software Engineering]: Software Architectures—Patterns

General Terms

Design

Keywords

Patterns, Interpattern Relationships

1. INTRODUCTION

A software pattern is a proven solution to a recurring problem that appears in the context of software development [1]. Describing the knowledge of experienced developers promotes sharing and reusing their knowledge. Many authors have published many patterns, and most of the patterns have relationships to other patterns. A pattern collection is a set of patterns that may or may not be related to each other.

In the design phase, developers break the system down gradually into components. Initially, the system has a higher abstraction level, and is independent from details in design and implementation. Near the end of development, the components have lower abstraction level, and depend on a concrete language and environment. There are software patterns addressing problems for each phase. The pattern to use for development must have an abstraction level matches the appropriate level in system development.

Developers should select patterns according to the development phase because a pattern that doesn’t matched the system’s abstraction level is not effective. Therefore, developers need to know the abstraction levels of patterns. In the basic design phase, developers should not be aware of idioms, and in the implementation phase, developers should not be aware of architectural patterns. The patterns mismatching current development phase may make developers confused. However, abstraction levels of patterns are different even if they belong to the same pattern catalog. Developers cannot clearly classify some patterns into architectural patterns, design patterns, or idioms.

For example, let’s consider Gang of Four (GoF)’s object-oriented design patterns [3] and PoSA’s patterns [1]. The GoF’s design patterns deal with the problems at the granularity of class design, and PoSA’s patterns deal problem at the granularity of system architecture design. However, for example, GoF’s Interpreter pattern is near an architectural pattern because it uses many other patterns directly and/or indirectly in its solution. It can be thought that the Model-View-Controller (MVC) pattern [4] [1] is near a design pattern because it treats individual applications. As in the above situations, it can be difficult for developers to select patterns fitting into considering level of abstraction. If there is a metric that position the Interpreter pattern near architectural patterns, developers can discuss whether to use it or not. However, actually, there is no objective metric capable to indicate that.

In this paper, we propose a metric that indicates the relative abstraction level of each pattern in a set of patterns. The proposed metric is based on the partially-ordered relationships between two patterns, and aims to assist in: understanding of the pattern’s abstraction level, classifying patterns, and selecting patterns to solve faced problems.

2. A METRIC MEASURING THE ABSTRACTION LEVEL OF DESIGN PATTERNS

There is a wide variation in software design abstraction level from architecture to detailed design to implementation. Developers focus on dividing a system into subsystems at
first. Next, they focus on the design of each subsystem, modules, and implementation. There can be software patterns in each abstraction level of software design.

Most patterns have one or more inter-pattern relationships. Authors of patterns often describe these relationships in the Related Patterns sections. Table 1 lists some examples of inter-pattern relationships. The Partially-ordered column shows whether each relationship is partially-ordered or not. For example, the structure of the Interpreter pattern’s syntax tree can be designed using the Composite pattern. That is a Uses relationship. Some inter-pattern relationships exist across different pattern catalogs. As another example, the MVC pattern uses the Observer pattern.

In this metric, we use only Uses, Refines, and Provides context relationships, i.e., the partially-ordered relationships shown on Table 1. The proposed metric has two principles below:

- Patterns that use other patterns, patterns which are more generalized, and patterns that are applied before other patterns have higher abstraction level.
- Patterns used by other patterns, patterns which are more specific, and patterns that are applied after other patterns have lower abstraction level.

We will present an intuitive explanation and a formal definition. Only the partially-ordered relationships affect the abstraction level. For instance, the Similar to relationship represents the existence of common properties between two patterns. Therefore, we use only the partially-ordered relationships to calculate the abstraction levels of patterns.

### 2.1 Intuitive explanation of the proposed metric

The reference count of a pattern is the total number of patterns that the pattern can reference transitively on the graph. The graph is composed of patterns (vertices) and inter-pattern relationships (edges). The backward reference count of a pattern is the total number of patterns that references the pattern transitively on the graph.

\[
P = \{p_1, p_2, \ldots, p_n\}, n \in \mathbb{N}.
\]

The partially-ordered relationship between a pattern \( p_1 \) and another pattern \( p_2, p_1, p_2 \in P \), is represented as \( \langle p_1, p_2 \rangle \). Therefore, the set of the relationships \( R \) is represented as

\[
R \subset P \times P.
\]

\((P, R)\) is a directed graph. \( R^+ \) is a transitive closure on \( R \). \( R^+ \) is defined as below:

\[
R_{i+1} = R^+ = \bigcup_{i=1}^{\infty} R^i.
\]

In addition, the set of patterns that can be retrieved transitively from a certain pattern \( p \) is a descendant of \( p \), represented as \( D(p) \). The set of patterns that a certain pattern \( p \) can be retrieved transitively are ancestors of \( p \), represented as \( A(p) \).

\[
D(p) = \{p_0 | p_0 \in P \land \langle p, p_0 \rangle \in R^+ \} \subset P
\]

\[
A(p) = \{p_0 | p_0 \in P \land \langle p_0, p \rangle \in R^+ \} \subset P.
\]

A pattern with many ancestors tends to be a part of other patterns. A pattern that has many descendants tends to use other patterns. Where \(|D(p)|\) denotes the number of patterns included in \( D(p) \), and \(|A(p)|\) denotes same of \( A(p) \), the abstraction level of a certain pattern \( p \) is defined as:

\[
a(p) = |D(p)| - |A(p)|.
\]
Since the metric depends on the relationships between patterns, the user should recompute the metric when the set of patterns are changed.

### 2.3 Visualization

Developers cannot understand intuitively abstraction levels only from numbers. In this section, we propose a visualization technique that uses the abstraction level.

In this technique, patterns with larger values for \( a(p) \) should be positioned above patterns with smaller values for \( a(p) \) should be positioned below. Only the vertical axis makes sense. In Figure 2, proposed visualization technique positions GoF’s design patterns. Ellipses indicate patterns and arrows indicate partially-ordered inter-pattern relationships. The abstraction level \( a(p) \) is used to position each pattern.

### 3. VISUAL DEMO OF THE METRIC

To experiment the metric and visualization techniques, we built a tool that calculates and displays the abstraction level of each pattern. The results are shown in Figure 2 and 3.

In Figure 2, we used “Uses”, “Refines” and “Provides context” relationships. The Interpreter and the Abstract Factory have high abstraction levels. The Interpreter references many other patterns, such as Visitor, Iterator and Composite, directly or indirectly. In the opposite side of that, the figure also shows that Prototype pattern and Singleton pattern have lower abstraction level. Actually, the Prototype and Singleton patterns are referred directly or indirectly by many other patterns. Using this proposed visualization technique, shown in Figure 2, developers can easily understand that some patterns are close to architectural patterns, other some patterns are close to idioms.

In Figure 3, we performed a similar analysis on GoF’s design patterns and PoSA’s patterns. Some architectural patterns, such as Layers and Pipes and Filters, are at the top of the figure. This means they have highest abstraction level in shown patterns. Interestingly, Interpreter pattern (a kind of design pattern) is positioned above Broker pattern (a kind of architectural pattern). In the middle of Figure 3, patterns belonging to each pattern catalog are mixed. Patterns can be connected if there are relationships of at least two patterns from each pattern catalog. The abstraction level and its visualization cannot be obtained without the proposed metric. Moreover, the developer can apply the proposed metric to a set of patterns, possibly from different catalogs, as we have illustrated.

### 4. RELATED WORK

Buschmann et al. proposed a classification of patterns: architectural patterns, design patterns, idioms [1]. This classification is based on abstraction level, however, we think it is too coarse-grained. Our metric can classify patterns into spectrum.

Martin proposed two metrics on packages [5], such as Affereent Couplings (Ca) and Efferent Couplings (Ce). Our metric and Martin’s OO-metrics is similar in the abstract structure of patterns/packages, however, our metric is for software patterns. Though our metric is defined as a simple subtraction, Martin’s metrics use a more complex calculation.

Cutumisu et al. have proposed four metrics for pattern catalogs: usage, coverage, utility and precision [2]. Those metrics use the number of patterns in a pattern catalog and the number of adapted/unadapted instances of patterns. In contrast, our metric uses partially-ordered inter-pattern relationships.

There are researches about inter-pattern relationships [8, 6, 7, 1]. Noble has surveyed inter-pattern relationships and has roughly classified into three categories, such as Use, Refine, Conflict. The Use and Refine relationships are partially-ordered, but the Conflict relationship is unordered. Since our metric is based on partially-ordered relationships, the metric works on Use and Refine relationships.

### 5. CONCLUSION

In this paper, we proposed a metric to measure the abstraction level of a pattern, based on partially-ordered inter-pattern relationships. The advantages and disadvantages of the proposed metric are the following:

**Advantages:**

- It can classify patterns into spectrum.
- It is simple and easy to understand.
- It can be applied to different sets of patterns.
- It can be used to understand the relationships between patterns.

**Disadvantages:**

- It may not be able to classify patterns with very high or very low abstraction levels.
- It may not be able to capture all types of relationships between patterns.

---

![Figure 1: Inter-pattern relationships](image-url)
The metric can be applied across pattern catalogs because the method only uses inter-pattern relationships.

The metric scores can be used as a hint to position patterns in a pattern map.

Disadvantages:

- The metric score is a relative value, so it is not possible to compare scores from different sets of patterns.
- Before applying the metric, the user has to obtain inter-pattern relationships on the targeted set of patterns.

We plan to perform experiments to validate that our metric makes the user experience involved.

6. REFERENCES


Appendix

× means the direct product of two sets.
∧ means “and”.
∈ means inclusion of the left-side argument by the right-side argument.
∪[i=1] means the union of the following sets.
|S| means the number of elements included in set S.
Figure 3: GoF’s design patterns and PoSA patterns. vertical position depends on the score $a(p)$