Knowledge Discovery for Design Pattern Selection

Poonam Ponde[1](✉), Manisha Bharambe[2], Kavita Khobragade[3], and Manisha Suryawanshi[4]

[1] Nowrosjee Wadia College, Pune, India
poonamponde@nowrosjeewadiacollege.edu.in
[2] MES Abasaheb Garware College, Pune, India
mgb.agc@mespune.in
[3] Fergusson College, Pune, India
kavita.khobragade@fergusson.edu
[4] Modern College of Arts, Science and Commerce, Pune, India

Abstract. Design patterns are useful Software Engineering tools that enable the reuse of expert solutions to recurring problems. There are a large number of patterns, spread in multiple catalogs and in heterogeneous formats. Selecting and applying the right design pattern requires an in-depth understanding of patterns and their classification. The solution architects must either rely on the advice of experts or laboriously go through the available literature to find the relevant patterns. Pattern applicability will improve if the entire pattern knowledge is available in one place and in a standard format. If the pattern data is augmented with additional knowledge to guide the architect on choosing the right patterns for a particular requirement, it will be immensely useful and productive. The objective of the knowledge discovery process on the design pattern landscape is to extract useful relations and groups of patterns to enable users to select and apply patterns effectively. The present work discusses a model for analyzing existing pattern data, extracting knowledge thereof, and representing this knowledge in a format to enable pattern search and its application.

Keywords: Design Pattern · Knowledge Discovery · Clustering · Software Engineering

1 Introduction

Our society is becoming increasingly dependent on software intensive systems in all areas. Information systems today are large-scale, complex, and distributed. This complexity makes it necessary to design their architecture based on standard practices, Software Engineering principles, and expert knowledge. Design patterns are useful Software Engineering tools that enable the reuse of expert solutions to recurring problems [1, 2]. The concept of design patterns was introduced for computer science by Gamma et al., [1]. Subsequently, a large number of patterns have been developed to solve a variety of

S. Tamane et al. (Eds.): ICAMIDA 2022, ACSR 105, pp. 47–60, 2023.
https://doi.org/10.2991/978-94-6463-136-4_7

software problems. Applying the right design pattern requires an in-depth understanding of patterns and their classification. The solution architects must either rely on the advice of domain experts or laboriously go through the available literature to find the relevant patterns. Therefore, pattern selection is primarily done by experienced software engineers who have a deep knowledge of patterns. This is extremely hard for the novice [3].

If the entire pattern knowledge is available in a common pattern repository and in a standard format, it will help the architects. This still requires the architects to understand all patterns to make a selection. If the pattern data is augmented with additional knowledge to guide the architect on choosing the right patterns, it will be immensely useful and productive. This augmented knowledge is usually acquired from domain experts or practitioners. In the absence of such experts, the knowledge can be extracted from the pattern data itself by applying data mining tools and techniques.

This paper presents a novel approach for pattern selection by analyzing existing pattern data, extracting knowledge thereof, and representing this knowledge to enable pattern selection. The promising results of the work on security design patterns encourage the applicability of the proposed model for design patterns in any domain to aid pattern organization and selection.

2 The Pattern Selection Problem

The most crucial task in pattern-based software engineering is the selection of the right pattern to solve a particular problem. The pattern selection problem can be divided into two stages: *Pattern search* and *Pattern selection*. Pattern search means getting information about existing patterns from repositories, the literature or the Internet. Pattern selection involves choosing patterns from the set of patterns after the stage of pattern search. The simplified process [4] of selecting and applying patterns to solve a problem is shown in Fig. 1.

The process starts with the problem and its description. Based on the problem description, the pattern search is applied to find a set of patterns that can potentially solve the problem.

Pattern search requires significant effort and time for the following reasons:

1. The number of existing patterns is very large.

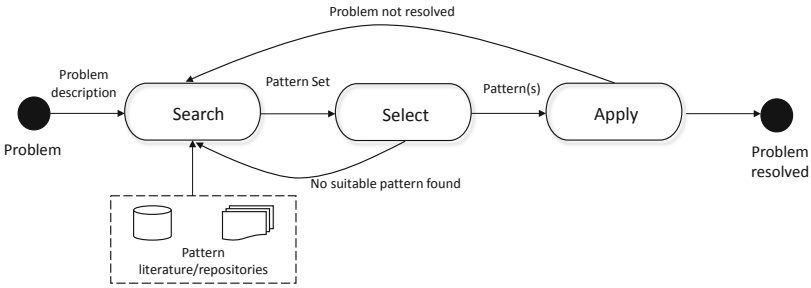

Fig. 1. The simplified process for pattern search and selection

2. The information of all patterns is not available at one place.
3. Pattern information is in a variety of formats including documents, images, HTML pages, research papers, and books.
4. All patterns are not described in a uniform manner.
5. The pattern landscape is complex and highly disorganized.

The pattern search yields multiple patterns. Not all these patterns give the solution to the problem. Pattern selection selects a pattern or patterns from the set of found/existing patterns. The last step is to apply the selected pattern in the software design to solve the problem. If the selected patterns do not solve the problem, the process is repeated.

Researchers have proposed various strategies and methods to solve the pattern selection problem which is discussed in the next section. This section also presents a brief background on design patterns and related work.

3 Background and Related Work

Christopher Alexander introduced patterns in the book, *A Pattern Language: towns, buildings, construction* [5]. Gamma et al., popularly known as the "Gang of Four" (GoF), first presented patterns for software design in their book '*Design Patterns: Elements of Reusable Object-Oriented Software'* [1]. As new technologies, computing paradigms, and architectural styles emerged, design patterns continued to evolve. The pattern landscape has been enriched by the addition of several patterns, pattern catalogues, pattern repositories, and pattern books. These include object oriented patterns [6], patterns for a specific domain such as patterns for enterprise application architecture [7], patterns for security [8, 9], J2EE web services patterns [10], patterns for context awareness [11, 12], patterns for embedded systems [13], IoT design patterns [14], Service Oriented Architecture patterns [15], Mobile design patterns [16], Cloud design patterns [17], Machine learning design patterns [18], patterns for digital platforms [19], AI systems [20] and many more. Analytical studies on the impact of applying design patterns on the software design, architecture and quality conclude that the software quality is greatly enhanced by applying the right patterns [21, 22].

Although there is a large volume of work on patterns themselves, the problem of pattern search and selection has not been addressed enough in literature. Various approaches to aid pattern selection are discussed here.

1. Online Pattern Repositories

To make pattern data more accessible various online pattern repositories were created. PatternForge [23] offers a centralized online location to control and manage design patterns and other software engineering instruments. PatternShare was a computer software design pattern web resource, hosted by Microsoft (upto 2009). Kienzle et al., created a repository of security patterns [24]. Open Pattern Repository [25] and Weiss and Birukou [26] have proposed the use of wiki's for building pattern repositories. The problem with repositories is that some repositories are not functional while others are not updated and maintained.

2. By classifying patterns

Efforts towards the organization of the design pattern landscape have led to pattern classification. A classification scheme identifies useful criteria based on which patterns can be classified. Some pattern classifications are based on the phases in the software development lifecycle by Buschmann et al. [2], Creational, structural and behavioral by Gamma et al. [1], According to the relationship between patterns [27], according to Software Design Level (Scalability) model [28], based on domain or technology: Microsoft patterns [29], J2EE/web services [10], Threat model, Core security principles etc. for security patterns [30, 31].

The problems with existing classification schemes are that most of them cover only a few patterns. Also, a pattern is not classified by all schemes.

3. Automated pattern selection and recommendation systems

The framework developed by Smith and Plante [32] detects design anti-patterns and makes dynamic recommendations to the programmers. Birukou et al. [33] present an implicit culture(IC) approach to select the appropriate patterns for a specific design problem. Dong et al. [34] proposed the DP-Miner tool to identify the design patterns based on the description of their structure including pattern matrix and weight. Kampffmeyer and Zschaler [35] formally described the GoF patterns by a Design Pattern Intent Ontology (DPIO) that is an extensible knowledge base of design patterns classified by their intent. Berghe et al. [36] use inductive logic programming techniques to choose the appropriate software patterns. Hasheminejad and Jalili [37] apply text classification on the patterns and the problem and then propose the best matching patterns from the classes. Hussain et al. [38] apply text categorization, feature selection and unsupervised learning techniques on the pattern descriptions for pattern recommendation.

Most of the approaches have been applied to only a few patterns or catalogs. Approaches based on Ontology or formal specification of patterns are highly complex [36]. Text categorization approaches require the pattern intent and description to be complete and in a standard format. In the following section, the knowledge discovery approach on pattern data is discussed.

4 Knowledge Discovery on Pattern Data

Knowledge discovery is a process that intelligently extracts valid and useful knowledge from raw information. The goal of the knowledge discovery process on patterns is to extract inherent inter-relationships between patterns and their attributes to aid pattern organization and selection. The knowledge discovery model proposed by Fayyad et al. [39] is one of the leading process models. The adoption of this model to the present work is shown in Fig. 2.

1. **Understanding the application domain**

There are many patterns for different domains. It is necessary to have a clear understanding of the domain for which pattern search is to be carried out.

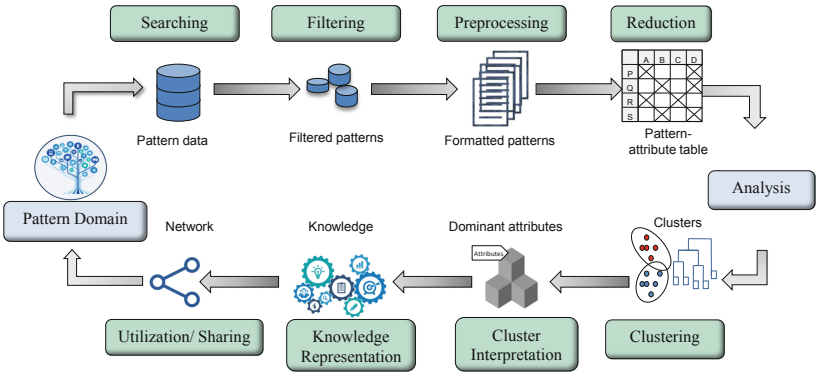

Fig. 2. Knowledge discovery model on pattern data

2. **Pattern search**

Pattern information is not available at one place. Pattern search involves collection of pattern information from various catalogs, repositories, books, research papers, and interaction with experts.

3. **Pattern filtering**

The pattern search yields multiple patterns. Not all patterns are useful in a particular context. Pattern filtering removes all unrelated and unwanted patterns.

4. **Preprocessing**

Pattern data collected from multiple sources must be collated, cleaned and preprocessed. To extract useful knowledge from patterns, patterns must be represented in a common format. The adoption of standard template for describing patterns will maintain uniformity and help improve the quality of patterns. This paper recommends the adoption of the following template by Buschmann et al. [2] (Table 1).

5. **Reduction: Pattern-attribute table**

The above pattern template is descriptive in nature. Pattern selection will be easier if the pattern information is augmented by a set of attributes. Several classification schemes exist which do the needful. However, there are overlaps in the existing schemes and all patterns are not classified using the same scheme. This paper encourages that a common set of domain-specific attributes must be applied to all the patterns. Pattern classification is represented as a two dimensional table of Patterns and Attributes (or criteria) as shown in Fig. 3. This table encapsulates the pattern knowledge and also captures the essence of the pattern domain.

6. **Pattern data mining**

To bring out hidden knowledge and inter-relationships between patterns, clustering is carried out. Clustering is an unsupervised data mining technique that groups similar

Table 1. Pattern description template

Element	Description
Name	The pattern name.
Context	Description of the circumstances in which the problem occurs.
Problem	Description of the problem arising in the context.
Forces	Description of the concerns and constraints that make the problem difficult to solve.
Solution	Description of the proven solution to the problem.
Also Known As	Aliases - other names for the pattern.
Example	A real world example which brings out the need for the pattern.
Structure	A specification of the composition of the pattern.
Dynamics	Description of the run-time behavior of the pattern.
Implementation	Description of how the pattern can be implemented.
Known Uses	Examples of pattern implementation in existing systems.
Consequences	The benefits and potential liabilities of using the pattern.
Variants	Description of specialized forms of the pattern
Related Patterns	A list of related patterns.
See Also	Resources to other patterns that address the similar problem.

	A1	A2	A3	A4
P1		✓		✓
P2	✓	✓		
P3	✓		✓	✓
P4			✓	
P5	✓	✓		✓

Fig. 3. Pattern-attribute table

objects without any prior knowledge of target classes [40]. Clustering will give a core set of pattern groups with related patterns. The extracted knowledge elements are in the form of clusters and their constituent patterns.

7. Cluster interpretation

Clusters contain a set of patterns that have some similarity. To identify the important attributes of the cluster, the concept of 'dominant cluster attributes' (having a percentage more than min-supp) is introduced by Ponde et al. [41]. The dominant attributes of a cluster reveal important characteristics of the cluster. These attributes play a crucial role in pattern selection.

8. **Knowledge representation**

The knowledge elements after the clustering process are: Patterns, Clusters and their patterns, and Dominant attributes of each cluster. These knowledge elements must be represented in a uniform format. A structured representation (such as XML) that is independent of underlying infrastructures will enable efficient storage and search operations.

9. **Utilizing and sharing the discovered knowledge**

The discovered knowledge is used for pattern selection in the following manner:

a. Identify attributes from the problem description.
b. Search the matching cluster having the problem attributes as dominant attributes.
c. Select patterns from the cluster to solve the problem.

5 Case Study: Knowledge Discovery on Security Patterns

Security is a critical requirement in software systems. There are more than 200 design patterns for security with multiple classification schemes [42]. This makes security design pattern selection a complex and time-consuming task. The proposed knowledge discovery model was applied on the security design patterns.

Pattern search yielded a large number of security patterns. For analysis, a filtered set of 211 patterns was used. Each pattern was classified by applying a core set of 23 classification attributes after careful consideration. These are:

– Lifecycle stages: Architectural, Requirement, Design, Implementation
– Core: Authentication, Authorization, Confidentiality, Availability, Non-Repudiation, Accountability, Key management
– Threat model: Spoofing, Tampering, Information Disclosure, Denial of Service, Elevation of privilege
– Trust boundary: Core, Perimeter, Exterior
– Type: Structural, Behavioral, Generic

The pattern-attribute table contains 211 rows and 23 columns. Each row represents a pattern with attributes as columns. To extract pattern relationships, clustering is carried out. The implementation is carried out in Python. The pattern-attribute table is represented in the.csv format for analysis. The Python code is:

```
import pandas as pd
dataset = pd.read_csv('PatternAttributeTable.csv')
X = dataset.iloc[:, 1:].values
```

To extract useful knowledge from this dataset, clustering is carried out. The clustering of unlabeled data can be performed with the Python module `sklearn.cluster`.. This

module contains various clustering methods such as AgglomerativeClustering, KMeans etc. Since the data is binary, the Agglomerative Hierarchical clustering algorithm is applied. The clustering process requires the selection of Linkage method and Distance metric [40].

The commonly used linkage methods are:

1. Single Linkage: Gives shortest distance between a pair of observations i and j in two clusters A and B.

$$\mathrm{L(A,\ B)\ =\ min(D(i,\ j)),\ i \in A,\ j \in B} \tag{1}$$

2. Complete Linkage: Gives the maximum distance between a pair of observations i and j in two clusters A and B.

$$\mathrm{L(A,\ B)\ =\ max(D(i,\ j)),\ i \in A,\ j \in B} \tag{2}$$

3. Average linkage: The sum of distances between each pair of observations in each cluster are divided by the number of pairs to get the average inter-cluster distance.

$$\mathrm{L(A, B)\ =} \frac{1}{(nA + nB)} \sum\nolimits_{i,j} D(i, j)\mathrm{i \in A,\ j \in B} \tag{3}$$

4. Ward linkage: This is similar to average linkage except it calculates the sum of squares of distances.

$$\mathrm{L(A, B)\ =} \frac{1}{(nA + nB)} \sum\nolimits_{i,j} D(i, j)^2\mathrm{i \in\ A, j \in B} \tag{4}$$

The various distance metrics are:

1. Euclidean: It is the most common distance measure which can be considered as the length of segment connecting two points using the Pythagorean theorem.

$$D(x, y) = \sqrt{\sum\nolimits_{i=1}^{n} (xi - yi)^2} \tag{5}$$

2. Manhattan: It is the sum of absolute differences between points.

$$D(x, y) = \sum\nolimits_{i=1}^{n} |xi - yi| \tag{6}$$

3. Minkowski: It is the generalized form of Euclidean and Manhattan distance.

$$D(x, y) = \left[\sum\nolimits_{i=1}^{n} |xi - yi|^p\right]^p \tag{7}$$

where, p is the norm.

Jaccard index: This is often used in applications with binary data. It measures the similarity between two sets of data to see which members are shared and distinct.

$$D(x, y) = 1 - \frac{x \cap y}{y \cup x} \tag{8}$$

Jaccard distance is recommended for binary data [40] and hence, the present work uses this metric. To choose the linkage method, the Cophenetic correlation coefficient is calculated. It gives the relation between raw data distances and dendrogram distances.

The `scipy.cluster.hierarchy`module provides the 'linkage' and 'cophenet' methods based on which the linkage and metric combination is chosen. The Jaccard metric with average linkage and Euclidean metric with Ward linkage gives the highest values i.e. 0.64.

The sample Python code for the same is:

```
from scipy.cluster.hierarchy import single, cophenet
from scipy.spatial.distance import pdist
z=sch.linkage(X, method='average', metric='jaccard')
cophenet(z, pdist(X))
```

The dendrogram obtained as a result of clustering is shown in Fig. 4. Clustering being an unsupervised technique, does not give any indication about the actual number of clusters. For hierarchical clustering, this number is obtained by observing the dendrogram and cutting the dendrogram to extract the clusters. The longest vertical line is cut to obtain 9 high level clusters. Each cluster further has its sub-clusters.

Constituent patterns are identified for each cluster. The defining attributes of each cluster are identified by calculating the percentage contribution of each attribute in the cluster. The attributes having a value > minsupp are the dominant attributes. They aid in security pattern selection for a specific security problem. A sample cluster with the dominant attributes and its patterns is shown in Table 2.

For pattern selection, the security attributes are first identified from the problem specification. The high level cluster having the dominant attributes = security attributes is first selected. If it is a high level cluster, then its sub-clusters are recursively searched till the desired patterns are found.

6 Validation of Results

It is necessary to validate the quality of the output after clustering. Cluster validation is a quantitative evaluation to assess the quality and reliability of clustering results. Cluster validation measures are of the following types:

i. Internal validation by evaluating the quality of clusters using only the data and without reference to external information.

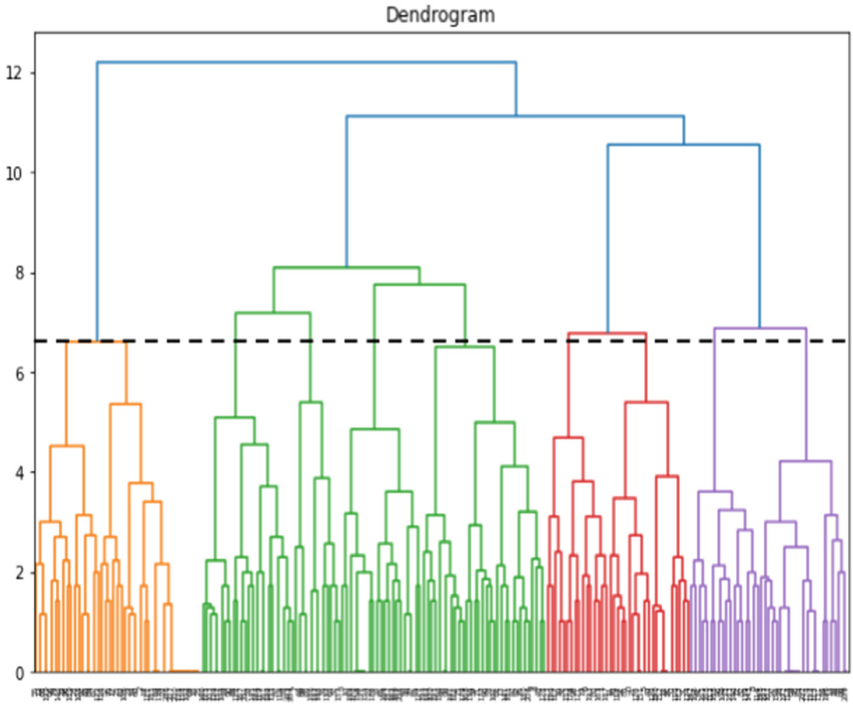

Fig. 4. Dendrogram

Table 2. Sample cluster details

Patterns	Dominant attributes				
	Lifecycle Stage	Quality Attribute	Threat	Trust Boundary	Type
12	Design	Availability	Denial of Service	Core	Generic
	58.33%	91.67%	100%	66.7%	83.33%
COMPARATOR CHECKED FAULT TOLERANT SYSTEM, LOAD BALANCER, TANDEM SYSTEM, TEST ON A STAGING SERVER, FAIL SECURELY, KEEP SESSION DATA IN CLIENT, KEEP SESSION DATA IN SERVER, PATCH PROACTIVELY, SAFE DATA STRUCTURE, SECURE ASSERTION, SESSION FAILOVER, SMALL PROCESSES					

ii. External validation measures the results of the clustering with externally supplied results such as class labels.

Since there are no class labels available, it is not possible to carry out external validation. Internal validation measures the "cohesion" and "separation" of clusters. Cohesion (Within-cluster Sum of Squares or WSS) measures how close the data points are within a cluster while Separation (Between-cluster Sum of Squares or BSS) measures

how distinct the clusters are from one another.

$$WSS = \sum_i \sum_{xCi} (x - m_i)^2 \tag{9}$$

$$BSS = \sum_i |Ci|((m - m_i)^2 \tag{10}$$

where |Ci| is the total number of points in cluster Ci, m_i is the centroid of Ci and m is the centroid of the data set. The Silhouette coefficient combines cohesion and separation. The Silhouette Coefficient s for a single sample is given as:

$$s = \frac{b - a}{\max(a, b)} \tag{11}$$

Here, 'a' is the mean distance between a sample and all other points in the same cluster and 'b' is the mean distance between a sample and all other points in the next nearest cluster. The Silhouette Coefficient for the dataset is the mean of the Silhouette Coefficient for each sample. The desired value is 1 and the worst value is −1. Values near 0 indicate overlapping clusters. Negative values indicate that a sample has been assigned to the wrong cluster.

The Python `sklearn.metrics.silhouette_score`gives the Silhouette coefficient for the given clustering. The sample code is:

```
  from sklearn import metrics
  metrics.silhouette_score(X, clustering.labels_, met-
ric='jaccard')
```

For the present work, the Silhouette coefficient value is 0.53797650221983363 which indicates fairly good clusters.

7 Conclusion

Design patterns present solutions to frequently occurring problems in software design. It is crucial to select and apply the right design patterns to solve a problem. The unorganized pattern landscape makes this task very complex. This paper presents a knowledge discovery approach to facilitate fast navigation through the pattern landscape and retrieve patterns which satisfy specific requirements. The model uses the process of knowledge discovery on the pattern data to extract useful relations and groups of patterns to enable users to select and apply patterns. The above model applied to security pattern data yields promising results in the form of pattern clusters and their dominant attributes which aid in pattern selection. As a future direction, the model will be applied on other design patterns.

References

1. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Object-Oriented Software. Addison Wesley (1994).
2. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns. Chichester, John Wiley & Sons (1996).
3. Sommerville, I.: Software Engineering. Addison-Wesley, Boston (2004).
4. Birukou. A.: A survey of existing approaches for pattern search and selection. In Proceedings of the 15th European Conference on Pattern Languages of Programs (EuroPLoP '10), Association for Computing Machinery, Article 2, pp. 1–13, New York, NY (2010)
5. Alexander, C.: The timeless way of building. Oxford University Press, New York (1979).
6. Beck, K., Cunningham, W.: Using Pattern Languages for Object-Oriented Program, Workshop on Specification and Design for Object-Oriented Programming (1987).
7. Fowler, M.: Patterns of Enterprise Application Architecture, Addison-Wesley Longman Publishing Co., Inc. (2002).
8. Yoder, J., Barcalow, J.: Architectural patterns for enabling application security. Proceedings of the Conference on Pattern Languages of Programs, pp. 1–31, Monticello/IL (1997).
9. Fernandez, E. B.: Security Patterns in Practice: Designing Secure Architectures Using Software Patterns. Wiley (2013).
10. Steel, C., Nagappan, R., Lai, R.: Core Security Patterns: Best Practices and Strategies for J2EE(TM), Web Services, and Identity Management. Prentice Hall International (2005).
11. Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., Steggles, P.: Towards a better understanding of context and context-awareness. In International symposium on handheld and ubiquitous computing (pp. 304–307). Springer, Berlin, Heidelberg (1999).
12. Riva, O., Di Flora, C., Russo, S., Raatikainen, K.: Unearthing design patterns to support context-awareness. In 4th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06), IEEE (2006).
13. White, E.: Making Embedded Systems: Design Patterns for Great Software. O'Reilly Media, Inc. (2011).
14. Washizaki, H., Yoshioka, N., Hazeyama, A., Kato, T., Kaiya, H., Ogata, S., Fernandez, E. B.: Landscape of IoT patterns. In 2019 IEEE/ACM 1st International Workshop on Software Engineering Research & Practices for the Internet of Things, pp. 57–60, IEEE (2019).
15. Erl, T.: SOA Design Patterns (paperback). Pearson Education (2008).
16. Neil, T.: Mobile design pattern gallery. O'Reilly Media, Inc. (2014).
17. Erl, T., Cope, R., Naserpour, A.: Cloud computing design patterns. Prentice Hall (2015).
18. Washizaki, H., Khomh, F., Guéhéneuc, Y. G., Takeuchi, H., Natori, N., Doi, T., Okuda, S.: Software-Engineering Design Patterns for Machine Learning Applications. Computer, 55(3), pp. 30-39 (2022).
19. Zaramenskikh, E., Oleynik, A.: Design Patterns for Digital Platforms. In Digital Transformation and New Challenges, pp. 109-123, Springer, Cham (2021).
20. Take, M., Alpers, S., Becker, C., Schreiber, C., Oberweis, A.: Software Design Patterns for AI-Systems. In EMISA, pp. 30–35 (2021).
21. Wedyan, F., Abufakher, S.: Impact of design patterns on software quality: a systematic literature review. IET Software, 14(1), pp. 1-17 (2020).
22. Ali, M., Elish, M.: A comparative literature survey of design patterns impact on software quality. In 2013 IEEE International conference on information science and applications (ICISA), pp. 1–7 (2013).
23. PatternForge home page, https://patternforge.org/, Last accessed 2022/10/04.
24. Kienzle, D. M., Elder, M. C., Tyree, D., Edwards-Hewitt, J.: Security patterns repository version 1.0. DARPA, Washington DC (2002).

25. Open Pattern Repository, http://www.iso-architecture.org/viewpoints/docs/opr-requirements-091007.pdf, Last accessed on 2022/10/01.
26. Weiss, M., Birukou, A.: Building a pattern repository: Benefitting from the open, lightweight, and participative nature of wikis. In International Symposium on Wikis (WikiSym), ACM, pp. 21–23 (2007).
27. Zimmer, W. Relationships between design patterns. Pattern languages of program design, 57, pp. 345-364 (1995).
28. Mowbray, T., Malveau, R.: CORBA design patterns. John Wiley & Sons, Inc. (1997).
29. Homer, A., Sharp, J., Brader, L., Narumoto, M., Swanson, T.: Cloud Design Patterns. Redmon, Washington, United States: Microsoft (2014).
30. Hafiz, M., Adamczyk, P., Johnson, R.: Towards an Organization of security patterns. IEEE Software, 24, pp. 52–60 (2007).
31. Washizaki, H., Fernandez, E, Maruyama, K., Kubo, A., Yoshioka, N.: Improving the classification of security patterns. Database and Expert Systems, pp. 165–170 (2009).
32. Smith, S. and Plante, D. R.: Dynamically Recommending Design Patterns. Proc. of the 24th International Conference on Software Engineering, pp. 499–504 (2012).
33. Birukou, A., Blanzieri, E., Giorgini, P.: Choosing the right design pattern: an implicit cultural approach. Technical Report, University of Trento, Italy (2006).
34. Dong, J., Lad, D. S., Zhao, Y.: DP-Miner: Design pattern discovery using matrix. In 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07), pp. 371–380, IEEE (2007).
35. Kampffmeyer, H., Zschaler, S.: Finding the Pattern You Need: The Design Pattern Intent Ontology. In: Model Driven Engineering Languages and Systems. pp. 211–225. No. 4735 in Lecture Notes in Computer Science (2007).
36. Berghe, A., Haaren, J. V., Baelen, S. V., Berbers, Y., Joosen, W.: Towards an automated pattern selection procedure in software models. Proc. of 22nd International conference on inductive logic programming (ILP 2012), pp. 68–73 (2012).
37. Hasheminejad, S., Jalili, S.: Design Patterns Selection: An Automatic Two Phase Method. The Journal of Systems and Software 85(2), pp. 408–424 (2012).
38. Hussain, S., Keung, J., Sohail, M., Khan, A., Ilahi, M.: Automated framework for classification and selection of software design patterns. Applied Soft Computing, 75, pp. 1-20 (2019).
39. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From data mining to knowledge discovery in databases. AI magazine, 17(3), 37 (1996).
40. Zaki, M. J., Meira Jr, W., Meira, W.: Data mining and analysis: fundamental concepts and algorithms. Cambridge University Press (2014).
41. Ponde, P., Shirwaikar, S., Kreiner, C.: An analytical study of security patterns. In Proceedings of the 21st European Conference on Pattern Languages of Programs, pp. 1–26 (2016).
42. Bunke, M.: Software-security patterns: degree of maturity. In Proceedings of the 20th European Conference on Pattern Languages of Programs, pp. 1–17 (2015).