Cyberpatterns: Towards a Pattern Oriented Study of Cyberspace

Hong Zhu

Department of Computing and Communication Technologies
Oxford Brookes University
Oxford OX33 1HX, UK
e-mail: hzhu@brookes.ac.uk

Abstract. A pattern represents a discernible regularity in the world or in manmade designs. In the prescriptive point of view, a pattern is a template from which instances can be created; while in the descriptive point of view, the elements of a pattern that repeat in a predictable manner can be observed and recognised. Similar to theories in sciences, patterns explain and predict regularities in a subject domain. In a complicated subject domain like cyberspace, there are usually a large number of patterns that each describes and predicts a subset of recurring phenomena, yet these patterns can interact with each other and be interrelated and composed with each other. The pattern-oriented research method studies a subject domain by identifying the patterns, classifying and categorising them, organising them into pattern languages, investigating the interactions between them, devising mechanisms and operations for detecting and predicting their occurrences, and facilitating their instantiations. This chapter illustrates this research methodology through a review of the research on software design patterns as an example of successful application of the methodology. It then discusses its possible applications to the research on cyberpatterns, i.e. patterns in cyberspace. It defines the scope of research, reviews the current state of art and identifies the key research questions.

1 Motivation

Since the first occurrence of the word cyberspace in 1980s in science fictions, cyberspace has expanded significantly and is expanding ever faster than before nowadays. During this period, the uses of the Internet, computer networks, and digital communication have been growing dramatically and the term “cyberspace” has been used to represent the new ideas and phenomena that have emerged. In this virtual space, individuals can interact, exchange ideas, share information, provide social support, conduct business, direct actions, create artistic media, play games, engage in political discussion, and so on, as a social experience. With the emerging of sensor networks, the Internet of things, mobile computing and cloud computing, cyberspace is now no longer isolated from the real world. The term

cyberspace has become a conventional means to describe anything associated with the Internet and the diverse Internet culture.

More importantly, the computational medium in cyberspace is an augmentation of the communication channel between real people. One of the core characteristics of cyberspace is that it offers an environment that consists of many participants with the ability to affect and influence each other. It is not simply a consensual hallucination experienced daily by billions of legitimate operators in every nation. In this space, children are being taught mathematics, natural and social sciences, and concepts and knowledge of all disciplines through, for example, Mass Open Online Courses. Social events that are big and small ranging from a family birthday party to a revolutionary protest are organised, for example, through social networks like YouTube and Facebook. Millions of business transactions are completed, employing e-commerce platforms like eBay and Amazon, etc. Yet, hacking for unauthorised access of computer systems, the spread of computer viruses and various forms of computer crimes also happen at every minute. Cyberspace is unthinkably complex and of great importance. The United States as well as many other countries have recognised the interconnected information technology operating across this space as part of the national critical infrastructure.

However, the enormous scale, the notorious complexity, the intangibility of the virtual world and its complicated relations to the real world have imposed grave challenges to the researchers to operate, manage, protect and further develop the this space. The notion of cyberpatterns (i.e. patterns in cyberspace) was thus proposed to meet such challenges as a new concept as well as a new research methodology for the study of cyberspace.

This chapter gives an introduction to the book by explaining the notion of cyberpatterns, defining its scope and domain, outlining the so-called pattern-oriented research method based on the notion of pattern, and discussing the key research questions to be answered in the study of cyberpatterns.

The chapter is organised as follows. Section 2 defines the notion of pattern, and Section 3 explains the pattern oriented research methodology, which is illustrated by a brief review of the research on software design patterns as an example of a successful application of the methodology. Section 4 explores the potential applications of pattern-oriented research methods to the study of cyberspace and related subject areas.

2 The Notion of Pattern

Pattern is a word widely uses in many subject domains of computer science and technology as well as many other scientific and technological disciplines. For example, in artificial intelligence, pattern recognition is an active and fruitful research topic in image processing and machine learning. In software engineering, research on design patterns has been an active research topic since the 1990s.

It aims at systematically representing codified and reusable design knowledge. It has been extended into research on requirement analysis patterns, process patterns, testing patterns, and so forth. The notion of software design patterns are in fact borrowed from a theory of architectural designs proposed by Alexander [2] in the 1970s.

Generally speaking, a pattern means a template from which instances can be created. Such a view of pattern is called the *prescriptive view*, because it provides a means for generating the instances of the pattern. Natural patterns such as spirals, meanders, waves, foams, tilings, cracks, etc., can be created by operations such as symmetries of rotation and reflection.

Another core characteristic of the notion of pattern is that a pattern represents a discernible regularity in the world or in a manmade design. As such, the elements of a pattern repeat in a predictable manner. Not only can such repetition be generated using operators as in the prescriptive view, but also be observed and recognised. For patterns in nature and art, the regularity can often be directly observed by any of the five senses. But patterns in cyberspace are more like the abstract patterns in science, mathematics, or language, which may be observable only by analysis. This leads to the *descriptive view* of patterns, which is an alternative to the prescriptive view. In this view, a pattern describes the instances of the pattern, thus providing a means to recognise whether an element in the domain is an instance of the pattern.

Nature patterns are often chaotic, never exactly repeating, and often involve fractals. This means that the predictions can be less accurate as one may expect. However, all such patterns have an underlying mathematical structure through which regularities manifest themselves as mathematical theories and functions. Therefore, scientific experiments can be carried out to demonstrate the regularity that a pattern predicts to validate the correctness of the documentation and specification of the pattern. This is similar to the sciences, where theories explain and predict regularities in the world. However, as one of the main differences from science theories, a subject domain may have a large number of patterns that each describes and predicts one subset of phenomena, yet these patterns can interact with each other and be interrelated and composed with each other. Such a set of validated patterns formulates a scientific foundation for the study of a space of phenomena. Thus, patterns are classified, categorised, and organised into pattern languages through assigning a vocabulary to the patterns in a subject domain.

3 Pattern-Oriented Research Methodology

These general features of patterns suggest that the notion of pattern is useful to investigate on the following types of research questions:

- finding the regularities of the complicated phenomena in a subject domain like cyberspace,

- understanding such repeating phenomena through discovery of the underlying mathematical structures,
- facilitating the observations of the phenomena in order to detecting the occurrences of a repeating phenomena,
- defining the operations to detect, predict and prevent and/or reproduce such phenomena,
- determining the relationships between various repeating phenomena,
- devising the mechanisms for classifying and categorising such phenomena,
- standardising the vocabulary of such phenomena to form pattern languages, etc.

Systematically studying all aspects of patterns in a subject domain is a research methodology that we believe is suitable for the research of cyberspace's related subjects. This methodology is called *pattern-oriented* in the sequel.

To illustrate how this research methodology works, this section briefly reviews the research on software design patterns. It is perhaps the most advanced subject domain in which pattern-oriented research methodology is applied.

3.1 Software Design Patterns: An Example of Applying the Pattern-Oriented Research Method

Software, as the most complex manmade artefact, is difficult to design and implement. There are a great number of software systems that have been developed for a vast range of different application domains. Some are very successful; while some others completely fail. A key research question is therefore:

- *Can design knowledge contained in the successful software systems be extracted and reused in the design of other systems?*

The pattern-oriented research methodology answers this question by regarding such successful design knowledge as patterns. In particular, design patterns are regarded as codified reusable solutions to recurring design problems [18, 3]. In the past two decades, much research on software design patterns has been reported in the literature. The following briefly reviews the existing work on software design patterns to illustrate how the pattern-oriented research method works.

3.1.1 Identification and documentation of design patterns

Design patterns are mostly identified by experienced software designers manually by recognising the recurring design problems and the successful examples of design solutions. These samples of successful solutions are abstracted into a template so that it can be applied to a wide range of applications but of the same design problem. Typically, such a template of design solutions consists of a number of software components, called *participants* of the solution, and they are connected to each other in certain specific configurations. The instantiation of the template is also illustrated by some typical examples. The applicability of the pattern, alternative

designs and relations to other patterns are also made explicit as a part of the pattern.

Usually, such a pattern is documented in the so-called *Alexandrian format*. In this format, design principles are first explained in informal English, and then clarified with illustrative diagrams and specific code examples [18]. The diagrams serve as the template and show how the participants interact with each other and the configuration of the connections. The specific code examples show the typical instances of the pattern. This format is informative for humans to understand the design principle underlying the pattern and to learn how to apply patterns to solve their own design problems.

3.1.2 Catalogues of patterns and pattern languages

Design patterns are not only identified and documented in isolation, but also systematically categorised and their relationships charted. Among many types of design patterns that have been studied are object-oriented program designs [18, 20, 21, 17], enterprise system designs [3, 22, 24, 42, 37], real-time and distributed systems [42, 10, 16], fault tolerance architectures [23], pattern-oriented software architectures [9, 8], and program code for implementing security feature [36, 37], etc.

The most important relationships between design patterns include:

- *Sub-pattern*: Pattern *A* is a sub-pattern of pattern *B* means that every instances of pattern *A* is also an instance of pattern *B*.
- *Uses:* Pattern *A* uses pattern *B* means that certain participants (i.e. the components) in pattern *A* is designed and implemented by using pattern *B*.
- *Composition:* Pattern *A* is a composition of pattern *B, C* and so on, if pattern *A* is formed by put pattern *B* and *C* etc. together.
- *Alternative Choice:* Pattern *A* and pattern *B* are alternative choices, if they serve certain common design goals and solve some common design problems, but they also have some different properties thus used in difference situations.

The systematic categorising of design patterns has led to a set of vocabulary of design terminology, which forms the so-called pattern languages.

3.1.3 Formal specification of design patterns

The documentation of design patterns in Alexandrian format is informal and hence brings such ambiguity that it is often a matter of dispute whether an implementation conforms to a pattern or not. Empirical studies show that poor documentation of patterns can lead to poor system quality, and can actually impede software maintenance and evolution [26]. Mathematical notations are thus employed to help eliminate this ambiguity by clarifying the underlying notions. Several formalisms for formally specifying OO design patterns have been advanced. [1, 32, 39, 19, 6]

In spite of differences in these formalisms, the underlying ideas are quite similar. That is, patterns are specified by constraints on what are its valid instances via defining their structural features and sometimes their behavioural

features too. The structural constraints are typically assertions that certain types of components exist and have a certain configuration of the structure. The behavioural constraints, on the other hand, detail the temporal order of messages exchanged between the components.

Formally, a design pattern P can be defined abstractly as a tuple $< V, Pr_S, Pr_D >$, where $V = \{v_1 : T_1, \cdots, v_n : T_n\}$ declares the components in the pattern, while Pr_S and Pr_D are predicates that specify the structural and behavioural features of the pattern, respectively. Here, v_i in V is a variable that ranges over the type T_i of software elements, such as class, method, and attribute. The predicates are constructed from primitive predicates either manually defined, or systematically induced from the meta-model of software design models. The semantics of a specification is the ground formula $\exists V \cdot (Pr_S \wedge Pr_D)$. Such a formalism for the specification of design patterns enables the conformation of a design or an implementation of a software system to a pattern be formally verified and automatically detected. This is in fact a descriptive view of design patterns.

An alternative approach to the formal specification of design patterns is the so-called transformational approach. In this approach, a design pattern is defines as a transformation rule that when applied to a flawed design (or implementation) results in a new design (or implementation) that conforms to the pattern [30]. Such a specification of pattern is prescriptive.

3.1.4 Development of software tools

With the documentation of design patterns, a large number of software tools have been developed to support various uses of design patterns in software engineering, which include

- Detection of design patterns in program code for reverse engineering [33, 25, 34, 7, 31, 15] and in designs represented as UML models [27, 28, 29, 44, 45]. Such tools are useful to reverse engineering.
- Instantiation of patterns for software design. Such tools are employed as parts of modelling tools and included as plug-ins in integrated software development environments, such as Eclipse, NetBeans, etc.
- Visual and graphical representation of patterns. For example, Dong et al. implemented a tool [14] for computer-aided visualisation of the application of design patterns in UML class diagrams. Their tool, deployed as a web service, identifies pattern applications, and does so by displaying stereotypes, tagged values, and constraints. Their experiments show that the tool reduces the information overload faced by designers.

3.1.5 Composition of design patterns

Although each pattern is documented and specified separately, they are usually to be found composed with each other with overlaps except in trivial cases [35]. Thus, pattern composition plays a crucial role in the effective use of design knowledge.

Composition of design patterns has been studied informally by many authors, for example, in [8, 35]. Visual notations, such as the *Pattern:Role* annotation and a

forebear based on Venn diagrams, have been proposed by Vlissides [41] and widely used in practice. They indicate where, in a design, patterns have been applied so that their compositions are comprehensible.

Pattern compositions have also been studied formally through formalisation and extension of the *Pattern:Role* graphic notation by considering pattern compositions as overlaps between pattern applications [11, 13, 12, 40, 38, 4]. A set of six operators on design patterns were defined with which the composition and instantiation of design patterns can be accurately and precisely expressed [5]. The algebraic laws of these pattern operators were also studied so that the equivalence of two pattern composition and instantiation expressions can be proved via equational reasoning [43].

Pattern decomposition, i.e. to work out how a design or a pattern is composed from a set of known patterns, is of particular importance in software engineering, especially in software reverse engineering. This is achieved through recognising the instances of patterns in the program code or in the design model represented in UML.

In summary, software design is a highly complicated domain. The research on design patterns started with the definition of the notion of design patterns as repeating phenomena in this domain, i.e. the solutions of recurring design problems. This notion of pattern is consolidated by the identification of certain design patterns and demonstration of their uses in software design. The research further developed with such a notion of pattern as the central concept to answer various research questions like how to document and specify design patterns, how patterns relate to each other and interact with each other, how to detect patterns when given a design or implementation, how find applicable design patterns and to instantiate a design pattern when given a design problem, etc.

3.2 Advantages and Benefits

The fruitful research on software design patterns and its wide applications in software development shows the following advantages and benefits of pattern-oriented research method.

1. Patterns represent domain knowledge in fine granularity, in which the complicated knowledge of the subject domain (such as software design) is divided into self-contained composable pieces.
2. Patterns, being self-contained, can be described and documented with high readability and learnability.
3. Each pattern as a piece of domain knowledge can also be easily validated and formally specified relatively independent of the other patterns.
4. Patterns codify domain knowledge in such a way that each pattern's applicability can be easily recognised and they can be flexibly combined through pattern composition.
5. The relationships between the patterns and the way patterns are composed and combined are also important parts of the domain knowledge, which can also be formally define, specified, and applied.

6. Automated tools can be developed and employed for various uses of the knowledge.
7. A complete knowledge of the subject domain can be developed incrementally by gradually identifying more and more patterns and the whole picture of the domain knowledge emerges in the form of catalogues of the patterns when a sufficient number of patterns are identified. Meanwhile, the incompleteness of a set of patterns does not affect the usefulness of the known knowledge.

In the next section, we discuss how the pattern-oriented research methodology exemplified above can be applied to the research on cyberspace.

4 Patterns in Cyberspace

The notion of cyberpattern advocates the application of the pattern-oriented research methodology to the study of cyberspace. Here, we discuss the scope of the research, the core research questions to be addressed as a road map for the future research.

4.1 What are patterns in cyberspace?

The first key research questions to be addressed in pattern-oriented research is:

- *What are the patterns in cyberspace?*

As discussed previously, cyberspace is a complicated subject domain. There is a vast range of phenomena that can be observed, recorded and detected as patterns. The following are just some examples of the patterns that have already been studied by researchers in various research communities.

- *User behaviour patterns*: Users of particular resources and applications on the internet tend to demonstrate certain regularities. Such regularities can be described as patterns, detected if abnormal behaviour occurs, for example, for intruder detection.
- *Workload patterns*: The workload on a particular server or a particular application on the internet varies, but often demonstrates certain pattern of variation, for example, due to the common behaviour of the users. Such patterns can be used to improve system performance and efficiency through load balancing and power management of the cluster.
- *Network traffic patterns*: The communication traffic on a particular network also exhibit clear patterns of variation according to the time and date as the users of the network have common behaviour patterns. Such patterns are useful for network optimisation.
- *Social network patterns*: People connecting with each other through social networks demonstrate certain patterns, for example, to form a network that observes the so-called Small World and Power Law of scale-free networks. The information propagates through such a social network in certain patterns, too.

- *Attack patterns*: The hackers attack the resource on the internet use certain information techniques. Thus, their behaviours also have certain dynamic characteristics that repeat inevitably.
- *Security design patterns*: To combat malicious attacks over the internet to attack computer facilities and resources, online systems employ security facilities to prevent, detect and mitigate security attacks. The vast range of such security facilities and mechanisms has certain common structures and implementations that can be represented as patterns like software design patterns.
- *Patterns of vulnerabilities*: The security facilities and mechanisms employed to protect internet resources often have weakness and vulnerabilities. Efforts have been made in the research on cybersecurity to identify, classify and categorise such vulnerabilities. Each type of vulnerability is in fact a pattern although they are not called so in the literature.
- *Digital forensic patterns*: In the investigation of cybercrime and more generally digital computer crimes, common investigation processes and guidelines have been developed for various types of crimes. They can be regarded as patterns, too.

Further research should be directed to answer questions like:

- *How can various types of cyberpatterns be identified?*
- *How should various types of cyberpatterns be described, documented and specified?*
- *How can cyberpatterns of various types be classified and catalogued systematically?*

The standards for the documentation and specification of various types of cyberpatterns are of particular importance to answer the following research questions:

- *How can various types of cyberpatterns be stored, retrieved and applied with the support of computer software tools and used automatically?*

4.2 How do cyberpatterns interrelate and interact with each other?

The pattern-oriented research methodology not only studies patterns individually, but also the relationship between them and the interaction they may have. A key research question to be addressed is therefore:

- *What are the relationships between various types of cyberpatterns?*
- *How cyberpatterns of the same type and across different types interact and interrelate with each other?*

For each type of patterns, one pattern may be related to another as its sub-pattern, and one is composed of some others, etc., as in the design patterns. In addition to such relationships, patterns in cyberspace of different types may also be related to each other in more complicated forms. Figure 1 below shows some ideas about the relationships between the types of patterns.

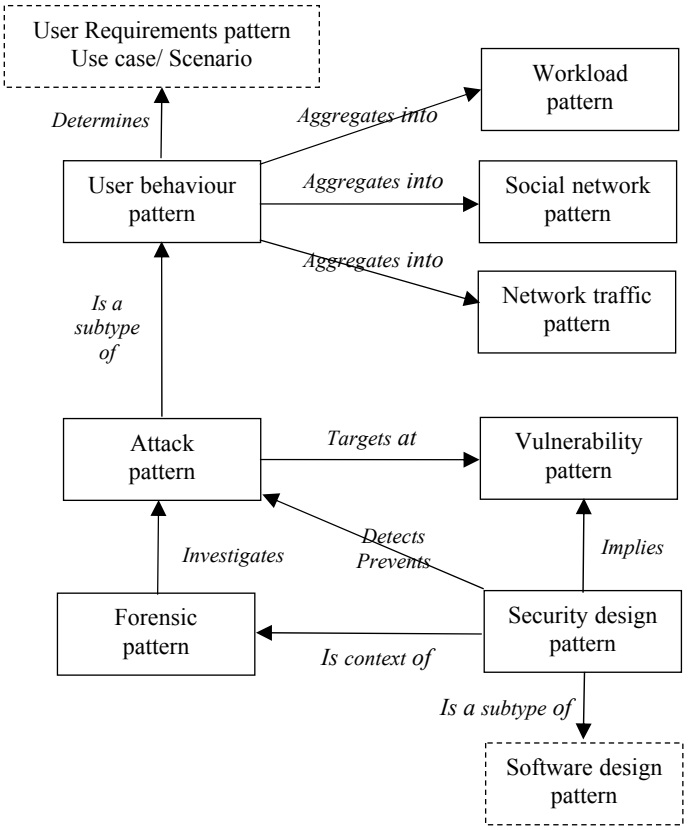

Fig. 1. Potential relationships between different types of cyberpatterns

Understanding these relationships will significantly improve our understanding of cyberspace and the effectiveness of using and protecting the infrastructure and resources on the Internet.

For example, consider the relationships between security design patterns and attack patterns. Security designs are often made by composing a number of design patterns that each pattern protects against one or more type of attack patterns. As illustrated in Figure 2, a problem that worth studying is to prove that *the composition of security design patterns can protect against the composition of attack patterns.*

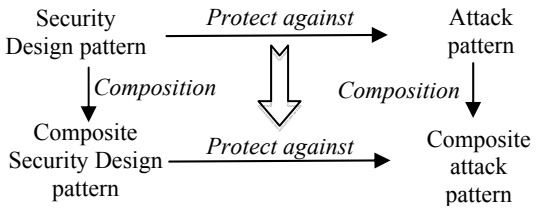

Fig. 2 Potential relationships between different types of cyberpatterns

5 Conclusion

A pattern represents a discernible regularity in a subject domain. It is a template from which instances can be created. It represents knowledge of recurring phenomena that can be observed and recognised. Similar to theories in sciences, patterns explain and predict regularities in certain phenomena. In a complicated subject domain like cyberspace, there are usually a large number of patterns that each describes and predicts a subset of recurring phenomena, yet these patterns can interact with each other and be interrelated and composed with each other.

The pattern oriented research method studies a subject domain by identifying the patterns, classifying and categorising them, organising them into pattern languages, investigating the interactions between them, devising mechanisms and operations for detecting and predicting their occurrences, and facilitating their instantiations. It is applicable to complicated subject domain like cyberspace.

References

1. Alencar, P.S.C., Cowan, D.D., de Lucena, C.J.P.: A formal approach to architectural design patterns. In: M.C. Gaudel, J. Woodcock (eds.) Proceedings of the Third International Symposium of Formal Methods Europe on Industrial Benefit and Advances in Formal Methods (FME'96), Lecture Notes In Computer Science, pp.576 – 594. Springer-Verlag (1996)
2. Alexander, C.: A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York (1977)
3. Alur, D., Crupi, J., Malks, D.: Core J2EE Patterns: Best Practices and Design Strategies, 2nd edn. Prentice Hall (2003)
4. Bayley, I., Zhu, H.: On the composition of design patterns. In: Proceedings of the Eighth International Conference on Quality Software (QSIC 2008), pp. 27–36. IEEE Computer Society, Oxford, UK (2008)
5. Bayley, I., Zhu, H.: A formal language of pattern composition. In: Proceedings of The 2nd International Conference on Pervasive Patterns (PATTERNS 2010), pp. 1–6. XPS (Xpert Publishing Services), Lisbon, Portugal (2010)

6. Bayley, I., Zhu, H.: Formal specification of the variants and behavioural features of design patterns. Journal of Systems and Software **83**(2), 209–221 (2010)
7. Blewitt, A., Bundy, A., Stark, I.: Automatic verification of design patterns in Java. In: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), pp. 224–232. ACM Press, Long Beach, California, USA (2005). URL http://www.inf.ed.ac.uk/ stark/autvdp.html
8. Buschmann, F., Henney, K., Schmidt, D.C.: Pattern-Oriented Software Architecture: On Pat- terns and Pattern Languages, vol. 5. John Wiley & Sons Ltd. (2007)
9. Buschmann, F., Henney, K., Schmidt, D.C.: Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing, vol. 4. John Wiley & Sons Ltd., West Sussex, England (2007)
10. DiPippo, L., Gill, C.D.: Design Patterns for Distributed Real-Time Systems. Springer-Verlag, New York, Inc., Secaucus, NJ, USA (2005)
11. Dong, J., Alencar, P.S., Cowan, D.D.: Ensuring structure and behavior correctness in design composition. In: Proceedings of the IEEE 7th Annual International Conference and Workshop on Engineering Computer Based Systems (ECBS 2000), pp. 279–287. IEEE CS Press, Edinburgh, Scotland (2000)
12. Dong, J., Alencar, P.S., Cowan, D.D.: A behavioral analysis and verification approach to pattern-based design composition. Software and Systems Modeling **3**, 262–272 (2004)
13. Dong, J., Alencar, P.S.C., Cowan, D.D.: Correct composition of design components. In: Proceedings of the 4th International Workshop on Component-Oriented Programming in conjunction with ECOOP99 (1999)
14. Dong, J., Yang, S., Zhang, K.: Visualizing design patterns in their applications and compositions. IEEE Transactions on Software Engineering 33(7), 433–453 (2007)
15. Dong, J., Zhao, Y., Peng, T.: Architecture and design pattern discovery techniques - a review. In: H.R. Arabnia, H. Reza (eds.) Proceedings of the 2007 International Conference on Software Engineering Research and Practice (SERP 2007), vol. II, pp. 621–627. CSREA Press, Las Vegas Nevada, USA (2007)
16. Douglass, B.P.: Real Time Design Patterns: Robust Scalable Architecture for Real-time Systems. Addison Wesley, Boston, USA (2002)
17. Fowler, M.: Patterns of Enterprise Application Architecture. Addison Wesley, Boston, USA (2003)
18. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
19. Gasparis, E., Eden, A.H., Nicholson, J., Kazman, R.: The design navigator: charting Java programs. In: Proc. of ICSE'08, vol. Companion Volume, pp. 945–946 (2008)
20. Grand, M.: Patterns in Java, volume 2. John Wiley & Sons, Inc., New York, NY, USA (1999)
21. Grand, M.: Java Enterprise Design Patterns. John Wiley & Sons, Inc., New York, NY, USA (2002)
22. Grand, M.: Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, Volume 1. John Wiley & Sons, Inc., New York, NY, USA (2002)
23. Hanmer, R.S.: Patterns for Fault Tolerant Software. Wiley, West Sussex, England (2007)
24. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison Wesley, Boston, USA (2004)
25. Hou, D., Hoover, H.J.: Using SCL to specify and check design intent in source code. IEEE Transactions on Software Engineering 32(6), 404–423 (2006)
26. Khomh, F., Gueheneuc, Y.G.: Do design patterns impact software quality positively? In: Proceedings of the 12th European Conference on Software Maintenance and Reengineering (CSMR 2008), pp. 274–278. IEEE, Athens, Greece (2008)

27. Kim, D.K., Lu, L.: Inference of design pattern instances in UML models via logic programming. In: Proceedings of the 11th International Conference on Engineering of Complex Computer Systems (ICECCS 2006), pp. 47–56. IEEE Computer Society, Stanford, California, USA (2006)
28. Kim, D.K., Shen, W.: An approach to evaluating structural pattern conformance of UML models. In: Proceedings of the 2007 ACM Symposium on Applied Computing (SAC'07), pp.1404–1408. ACM Press, Seoul, Korea (2007)
29. Kim, D.K., Shen, W.: Evaluating pattern conformance of UML models: a divide-and-conquer approach and case studies. Software Quality Journal 16(3), 329–359 (2008)
30. Lano, K., Bicarregui, J.C., Goldsack, S.: Formalising design patterns. In: BCS-FACS Northern Formal Methods Workshop, Ilkley, UK (1996)
31. Mapelsden, D., Hosking, J., Grundy, J.: Design pattern modelling and instantiation using DPML. In: CRPIT '02: Proceedings of the Fortieth International Conference on Tools Pacific, pp. 3–11. Australian Computer Society, Inc. (2002)
32. Mikkonen, T.: Formalizing design patterns. In: Proc. of ICSE'98, Kyoto, Japan, pp. 115–124. IEEE CS (1998)
33. Niere, J., Schafer, W., Wadsack, J.P., Wendehals, L., Welsh, J.: Towards pattern-based design recovery. In: Proceedings of the 22nd International Conference on Software Engineering (ICSE 2002), pp. 338–348. IEEE CS, Orlando, Florida, USA (2002)
34. Nija Shi, N., Olsson, R.: Reverse engineering of design patterns from java source code. In: Proc. of ASE'06, Tokyo, Japan, pp. 123–134. IEEE Computer Society (2006)
35. Riehle, D.: Composite design patterns. In: Proceedings of the 1997 ACM SIGPLAN Conference On Object-Oriented Programming Systems, Languages and Applications (OOPSLA'97), pp. 218–228. ACM Press, Atlanta, Georgia (1997)
36. Schumacher, M., Fernandez, E., Hybertson, D., Buschmann, F.: Security Patterns: Integrating Security and Systems Engineering. John Wiley & Sons, West Sussex, England (2005)
37. Steel, C.: Applied J2EE Security Patterns: Architectural Patterns & Best Practices. Prentice Hall PTR, Upper Saddle River, NJ, USA (2005)
38. Taibi, T.: Formalising design patterns composition. Software, IEE Proceedings 153(3), 126–153 (2006)
39. Taibi, T., Check, D., Ngo, L.: Formal specification of design patterns - a balanced approach. Journal of Object Technology 2(4) (2003)
40. Taibi, T., Ngo, D.C.L.: Formal specification of design pattern combination using BPSL. Information and Software Technology **45**(3), 157–170 (2003)
41. Vlissides, J.: Notation, notation, notation. C++ Report (1998)
42. Voelter, M., Kircher, M., Zdun, U.: Remoting Patterns. John Wiley & Sons, West Sussex, England (2004)
43. Zhu, H., Bayley, I.: An algebra of design patterns. ACM Transactions on Software Engineering and Methodology 22(3), Article 23 (2013).
44. Zhu, H., Bayley, I., Shan, L., Amphlett, R.: Tool support for design pattern recognition at model level. In: Proc. of COMPSAC'09, pp. 228–233. IEEE Computer Society, Seattle, Washington, USA (2009)
45. Zhu, H., Shan, L., Bayley, I., Amphlett, R.: A formal descriptive semantics of UML and its applications. In: K. Lano (ed.) UML 2 Semantics and Applications. John Wiley & Sons, Inc. (2009). ISBN-13: 978-0470409084