Patterns have become part of the software development mainstream (see the sidebar “What Is a Pattern?” for a short definition). They’re available for all development phases, including analysis, documentation, design, testing, and configuration management, to name a few. In addition to documenting the current status of software patterns, our goal for this special issue is to reflect critically on and discuss the achievements of the pattern community.
While selecting the articles for this issue, we learned more about both the field’s achievements and misconceptions than we expected. Unfortunately, a significant number of submissions were based on misconceptions about patterns.

We condensed these misconceptions into three themes:

- **Patterns are applicable only in the design phase.** Many people know only design patterns—specifically, the patterns described in the Gang-of-Four book (*Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Addison-Wesley, 1994). However, currently, most new patterns are documented outside of software design.

- **Patterns are UML templates.** Many people notice only the solution structure when they look at a pattern—they tend not to look at the problem section, the forces, or the trade-offs. For them, the concrete UML diagram is the pattern. They fail to see that the participants depicted in the UML diagram are just roles, just an example solution structure; they don’t see that the pattern is also about interactions, forces, when to use the pattern, and what the consequences and trade-offs of using the pattern will be. Unfortunately, this often leads to using patterns in the wrong places. Even among the submissions to this special issue, many articles fell into the trap of conceiving patterns as UML templates. For us, this was reason enough to reject them.

- **Pattern application can be automated.** Based on the prior misconception, people tend to put patterns into UML tools (“Our tool supports patterns—all 23!”). Of course, this again misses the point. To find out whether and how to use a given pattern in a given context, tools that work with patterns would have to be able to semantically understand your design as well as the patterns’ trade-offs. Sure, it can be useful to generate some aspect of a pattern implementation, but this isn’t the important aspect of patterns. In today’s CASE tools, users must decide whether and how to use a pattern in their design. Tools merely automate some of the coding, which is arguably the least complex step in using a pattern.

To establish a common direction, the pattern community has found writers’ workshops highly valuable. At these workshops, which take place at the pattern conferences, patterns undergo massive review and discussion to ensure their correctness and usefulness. See “The Patterns Community” sidebar in this introduction for explanations of shepherding, writers’ workshops, and the pattern conferences.

This issue’s article “Past, Present, and Future Trends in Software Patterns,” by Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt, gives a brief history of patterns in the context of software development and discusses today’s software patterns landscape. You’ll learn how more and more individual patterns are grouped and organized into larger contexts such as pattern collections, systems, and languages. The article concludes by looking at the domains for which patterns will be documented in the future. We hope this motivates you to follow up on this vision and help to make it real.

To give a strong positive example of how to use patterns, “Using Patterns to Capture Architectural Decisions” by Neil B. Harrison, Paris Avgeriou, and Uwe Zdun shows how to understand and use patterns for building and documenting architectures. Using patterns in the context of software architecture is a proven yet somewhat unknown area today. The authors discuss the similarities between classical ways of documenting software architectures and using a pattern-oriented approach, which the authors prefer.

Now that we’ve seen how we can use patterns to document architecture and design knowledge, it becomes interesting to see how patterns are documented in the first place. Writing good patterns isn’t necessarily simple, and it’s not without pitfalls. Finding the right level of granularity and a suitable abstraction level...
when describing a given problem is a central challenge for pattern writers. In “Understanding the Power of Abstraction in Patterns,” Linda Rising shares her personal insights gained from documenting patterns in the area of change management. It’s exciting to see what kind of cross-relations you can discover while digging into the domain you derived your pattern from.

Every discipline, after a period of rapid expansion, needs some time for consolidation or it risks disintegration. The expansion of software patterns has produced a large body of work that now needs organization. “Organizing Security Patterns” by Munawar Hafiz, Paul Adamczyk, and Ralph E. Johnson documents early efforts to consolidate and organize a subset of software patterns in the security domain. Lessons learned through this process can help people trying to organize patterns for other domains.

We round out our selection of articles with “The Growing Divide in the Patterns World” by Dragos Manolescu, Wojtek Kozaczynski, Ade Miller, and Jason Hogg. They report on their study of patterns use in the software development industry. Based on this, they issue a call for action to the pattern community in order to bridge discovered gaps.

The final piece is a Point-Counterpoint discussion by Peter Sommerlad and James Noble on the pros and cons of design patterns as proposed in the Gang-of-Four book. We invite you to follow Sommerlad’s and Noble’s arguments about whether the original 23 design patterns lead to better or worse design.

We hope you find this special issue interesting and fun to read. If you want to learn more about patterns or become more involved with the community, consider going to one of the Pattern Languages of Programs conference.

Acknowledgments

We thank the reviewers for devoting their time and expertise to help us review all the submissions and select the articles in this issue.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.