Past, Present, and Future Trends in Software Patterns

Frank Buschmann
Siemens, Munich, Germany
frank.buschman@siemens.com

Kevlin Henney
Curbralan, Bristol, UK
kevlin@curbralan.com

Douglas C. Schmidt
Vanderbilt University, Nashville, USA
d.schmidt@vanderbilt.edu

Abstract

Software patterns have significantly influenced the way in which computing systems are designed and implemented during the past two decades, well above and beyond the most popular work in the field. This article discusses the past, present, and future trends of software patterns so that software developers and managers will understand where the field has come from, where it is headed, and where they can learn more about patterns to help their own projects.

Keywords: Software patterns, pattern languages, object-oriented frameworks.

1. Introduction

For more than a decade, patterns have influenced the way software architects and developers think about creating computing systems. Design-focused patterns provide a vocabulary for expressing architectural visions, as well as examples of representative designs and detailed implementations that are clear and concise. Presenting pieces of software in terms of their constituent patterns also allows developers to communicate more effectively, with greater conciseness and less ambiguity.

Since the mid 1990s many software systems, including major parts of the Java and .NET libraries and the development of many middleware platforms, were developed with the conscious awareness of patterns. Sometimes these patterns were applied selectively to address specific challenges and problems. Other times they were used holistically to help construct software systems starting with the initial definition of their baseline architectures to the final realization of their fine-grained details. The knowledge and conscious application of patterns is now a valuable commodity for software professionals.

Patterns have not always been as popular or pervasive as they are now. Although there were isolated publications on software patterns in the late 1980s and early 1990s, patterns entered the mainstream in 1994 with the publication of the "Gang of Four" book [1]. This book described 23 patterns derived largely from the authors' experience developing single-threaded, object-oriented, user-interface frameworks in Smalltalk and C++.

Despite being published over a decade ago, the Gang-of-Four remains the most popular and influential pattern work. Numerous books and papers have been published that address the Gang-of-Four patterns in various ways. Some publications are companions on how to implement the patterns in other programming languages, such as C# and Java. Other publications rework and integrate the Gang-of-Four patterns for specific application contexts, such as distributed computing, security, and XML processing.

Although the Gang-of-Four book has significantly influenced software design, much has changed since the mid 1990s. The technology landscape has shifted, approaches to software design have evolved and expanded, our understanding of development processes has matured, and we know more about patterns and how to document and apply them successfully to software development today. Ironically, the popularity and influence of the Gang-of-Four book is so pervasive that many software developers are unaware of how much the field has matured and where to find pattern publications that cover a broader range of domains and technologies than the Gang-of-Four book.

To help address this situation, this article discusses the past, present, and future trends of software patterns. The goal of the pattern community has long been to document and promote good software engineering practices. The goal of this article is to convey the breadth and depth of the patterns in practice so that software developers and managers will understand where the field has come from, where it is headed, where they can learn more about patterns to help their own projects, extending well beyond the scope of the seminal Gang-of-Four book.

2. A Brief History of Patterns for Software

In his 1999 keynote talk at the OOPSLA conference, philosopher of science James Burke noted that history rarely happens in the right order or at the right time, but the job of a historian is to make it appear as if it did. Likewise, this section summarizes key trends in the patterns literature during the past two decades, referencing relevant work during that period, though our list is far from exhaustive. More comprehensive references to the patterns literature is available at *http://hillside.net/patterns*.

2.1. Stand-alone Patterns and Pattern Collections

Inspired by the success of the Gang-of-Four book, much of the popular work on patterns in the mid- to late-1990s focused on *stand-alone patterns* and *pattern collections*. Stand-alone patterns are point solutions to relatively bounded problems that arise within specific contexts. Examples of popular stand-alone patterns include ITERATOR, which offers aggregate traversal without exposing aggregate representation details to callers, STRATEGY, which captures pluggable behavior, and WRAPPER FACADE, which encapsulates the functions and data provided by existing procedural APIs within more concise, robust, portable, and cohesive object-oriented interfaces.

Any significant software design inevitably includes many patterns, however, which means that a stand-alone pattern unusual in practice. The most obvious presentation of multiple patterns is in the form of a *pattern collection*. The most ambitious pattern collection to date is Grady Booch's ongoing work on the *Handbook of Software Architecture* (*http://www.booch.com/architecture*), which currently references ~2,000 patterns. Most pattern collections, including the Gang-of-Four book, are much more modest in size and ambition, however, often focusing on a particular kind of problem, context, or system, and numbering ~10s of patterns.

Many stand-alone patterns were initially created for—and honed by—writers' workshops at various "Pattern Language of Programming" (PLoP) conferences worldwide. For example, the PLoP and EuroPLoP pattern conferences in the USA and Europe have prospered and expanded to several new venues, including VikingPLoP in the Nordic countries, SugarLoafPLoP in Brazil, and KoalaPLoP in Australia. In a writers' workshop at a PLoP conference, authors read each others' patterns and discuss their pros and cons to help improve content and style. The most mature patterns from these various conferences can be found in the *Pattern Languages of Program Design* (PLoPD) book series, respective conference proceedings, online, or elaborated and incorporated into other books.

2.2. Pattern Relationships

Patterns that represent the foci for discussion, point solutions, or localized design ideas can be used in isolation with some degree of success. Patterns are generally gregarious, however, in that they form relationships with other patterns. Any given application or library will thus make use of many related patterns. Three of the most common types of pattern relationships include:

- *Patterns complements*, where one pattern provides the missing ingredient needed by another or where one pattern contrasts with another by providing an alternative solution to a related problem. The goal of cooperative complements is to make the resulting design more complete and balanced. For example, DISPOSAL METHOD complements FACTORY METHOD by addressing object destruction and creation, respectively, in the same design. This complementary combination supports designs that retain encapsulation of resource lifecycle policies, such as pooling. Patterns may also find themselves in competition, such as the use of BATCH METHOD to access the elements of an aggregate object without exposing its underlying implementation. This pattern provides an alternative to ITERATOR that is more suitable for access in distributed environments, where the cost of remote access makes the cost of a conventional ITERATOR prohibitive. A BATCH METHOD accesses the elements of an aggregate in bulk, reducing roundtrip network costs.

- *Pattern compounds* capture recurring subcommunities of patterns that are common and identifiable enough that they can be treated as a single decision in response to a recurring problem. For example, the coincident application of two patterns, such as a COMMAND implemented as a COMPOSITE, is so common it can be named naturally and almost without thought as a single entity, such as COMPOSITE COMMAND. Another example is BATCH ITERATOR, which brings together two complementary patterns, ITERATOR and BATCH METHOD, to address the problem of remotely accessing the elements of aggregates with large numbers of elements. A BATCH ITERATOR refines the position-based traversal of an ITERATOR with a BATCH METHOD for bulk access of many, but not all, elements.

- *Pattern sequences* generalize the progression of patterns and the way a design can be established by joining together predecessor patterns to form part of the context of each successive pattern. For example, [2] presents a pattern sequence for communication middleware that joins together the BROKER, LAYERS, WRAPPER FACADE, REACTOR, ACCEPTOR-CONNECTOR, HALF-SYNC/HALF-ASYNC, MONITOR OBJECT, STRATEGY, ABSTRACT FACTORY, and COMPONENT CONFIGURATOR patterns into a pattern sequence. A pattern sequence captures the unfolding of a design or situation, pattern by pattern, and such a progression can also be illustrated with a *pattern story*, which is a concrete example of application of a pattern sequence.

2.3. Pattern Languages

As early as 1995, leading authors in the pattern community began documenting groups of patterns for specific software development domains, particularly telecommunication systems. As shown in the first several PLoPD books, these patterns were more closely related than the stand-alone patterns and pattern collections published earlier. In fact, some patterns were so closely related that they did not exist in isolation. The patterns were organized into *pattern languages* in which each pattern built upon and wove together other patterns in the language. From one point of view, pattern languages can be seen as the logical extrapolation of the pattern relationships described in the previous subsection; from another point of view, the pattern relationships described in the previous subsection can be seen as constrained and simplified aspects and subsets of pattern languages.

Pattern languages aim to provide holistic support for using patterns to develop software for specific technical or application domains, such as e-commerce or communication middleware. They achieve this goal by enlisting multiple patterns for each problem that can arise in their respective domains and weaving them together to define a generative and domain-specific, pattern-oriented software development process.

Many pattern languages are elaborations and decompositions of stand-alone patterns and pattern sequences. It is only natural that these languages build upon, and recursively unfold and strengthen, the core qualities of their constituent patterns. A prominent example of such a pattern language appears in the book *Remoting Patterns*, which decomposes the BROKER pattern from [3] into a fully fledged description of modern communication middleware architectures. When BROKER was described as a stand-alone pattern it could not be that expressive, but the remoting pattern language managed to reveal this generativity.

2.4. Domains and Technologies Documented by Patterns

From the start, there has been an intimate relationship between patterns and frameworks, as evidenced by the many framework examples used to motivate patterns in the Gang-of-Four book. Experience has shown that mature frameworks exhibit high pattern density, making patterns an ideal descriptive tool for developing, evolving, and understanding frameworks. Beyond the interest in frameworks, the following domains and technologies are among those that have emerged as popular foci for pattern authors:

- *Distributed computing*. Distributed computing has been a popular focus for pattern authors for many years. For example, [2] presents an extensive pattern language for building distributed software systems. The language connects over 250 patterns that address topics ranging from defining and selecting an appropriate baseline architecture and communication infrastructure, to specifying component interfaces, their implementations, and their interactions. The patterns covered in the book also address key technical aspects of distributed computing, such as adaptation and extension, concurrency, database access, event handling, synchronization, and resource management.

- *Language- and domain-specific idioms*. Several programming styles have evolved and emerged over the past decade, including aspect-oriented software development, domain-driven design, model-driven software development, and generative programming. Each of these styles has its own patterns and best practices that distinguish programming in that style from other programming styles. Patterns and pattern collections have therefore been documented for these styles. Programming language idioms have been a common focus for published patterns, from Smalltalk to Python and from C++ to C#.

- *Fault tolerance and fault management*. As software is increasingly integrated into mission- and safety-critical systems there is a clear need for robust techniques to meet user dependability requirements. Patterns on fault tolerance and fault management have therefore been an active focus over the past decade. Several recent books [4] contain patterns and pattern languages that address fault tolerance and fault management for systems with stringent operational requirements.
- *Security*. Security has been a topic for patterns since the beginning of this millennium, mirroring the growing emphasis on security in software systems in general. [5] documents a range of patterns and pattern languages for security-related areas, such as authentication, authorization, integrity, and confidentiality. Many patterns in this book had been documented in earlier pattern publications, mostly in the proceedings of PLoP conferences. [6] is another book published in the area of networked software security.
- *Embedded systems*. A rapidly growing part of applications, such as pacemakers, controllers for power plants, and flight-critical avionics systems, embed intelligence in physical devices and systems,. Since these applications are inextricably connected to the physical environment, they must be designed to satisfy physical demands and limitations, such as dynamics, noise, power consumption, and physical size, in a timely manner. [7][8] document patterns for addressing these constraints without losing all the benefits of abstraction.
- *Process and organizational structure*. Much work on pattern languages has focused on improving software development processes and organizations over the years. Some recent books [9][10] address certain types of software development processes, such as distributed, agile, test-driven, and domain-driven software development, as well as software refactoring and reengineering. [11] integrates existing patterns on software development processes and organizations into an interconnected pattern language.
- *Education*. Patterns and pattern languages are popular vehicle for teachers and consultants to convey their knowledge on the art of teaching programming and software engineering. Some of the many publications in this area appear in [12], as well as the website of the 'pedagogical patterns project' (*http://www.pedagogicalpatterns.org*), which provides a forum for disseminating and discussing teaching patterns, with a focus on teaching effective software practice.

Some addition domains and technologies that we believe will be covered by future pattern authors appear in Section 4, *Where Patterns May Go Tomorrow*.

3. Where Patterns Are Now

In one sense, the original goal of the patterns community—to document and promote good software engineering practices—has been met to a significant extent. After more than a decade of experience in mining, documenting, and applying patterns, patterns are now established in mainstream software development practice. Patterns are used consciously in many production software projects and university curricula. Moreover, there is much more experience with the format used to document patterns, such as a deeper appreciation of the importance of capturing the forces that shape designs [13], compared with the format used in the Gang-of-Four book. We expect these trends to continue as developers increasingly understand the core patterns, as well as broaden their knowledge of the wealth of patterns and pattern languages available in the broader literature.

Although many software systems have succeeded by intentionally applying patterns, there have also been failures, due in large part to common misunderstandings about patterns, i.e., what they are and what they are not, what properties and purpose they have, their target audience, and the various pros and cons of using them. The pattern community has long been interested in understanding the underlying theories, forms, and methodologies of patterns, pattern languages, and their associated concepts to help codify knowledge about, understanding of, and effective application of, software patterns. The largest work in this area appears in [13], which integrates many facets of the pattern concept into a coherent whole.

Compared to the initial wave of pattern users following the publication of the Gang-of-Four book, software developers in general seem to have a better grasp of patterns than before, in terms of their experience using patterns effectively on software projects and their understanding of various aspects of the pattern concept. The quality of the published patterns and pattern languages has also generally increased. Most patterns and pattern languages published in the past several years are more expressive, more comprehensive, more precise, and more readable than many patterns and pattern languages published in the past.

4. Where Patterns May Go Tomorrow

Now that many software developers have some familiarity with the concept of patterns and how to apply patterns in their domains, that begs the question ‘What is there now left to say about patterns?’ We have been observing trends that shape the pattern literature and community, as well as forecasting the future of patterns every three to four since 1996 as part of our work on the *Pattern-Oriented Software Architecture* book series. This section presents our conjectures on future trends given the benefit of hindsight and our experience discovering, documenting, and applying patterns in practice.

We expect more technology- and domain-specific patterns and pattern languages will be documented. Not all technologies and domains of software development are yet addressed by patterns—and it may take decades to cover them all. In particular, patterns capture experience, and for newer domains and technologies this experience must first be gained, evaluated, and codified before effective patterns can be documented. The domains and technologies we predict will be addressed first include:

- *Service-Oriented Architecture (SOA)*. SOA is a style of organizing and utilizing distributed capabilities that may be controlled and owned by different organizations or groups. SOA is not a new concept, but it has become mainstream and ascended to buzzword status in recent years. The SOA approach builds upon many principles and technologies known from distributed computing and enterprise system integration, and can thus draw from a broad spectrum of existing patterns and pattern languages, such as those documented in [14][15]. Some SOA technologies, such as business process modeling, service orchestration, and ultra-large-scale systems, are still unexplored, however, and are not yet covered by corresponding documented patterns.

- *Generative software technologies*. The vast majority of patterns documented since the early 1990s have been mined from object-oriented software written in third-generation programming languages. Patterns and pattern languages, however, have already started to influence and support other software development approaches, in particular aspect-oriented software development (AOSD) and model-driven software development (MDSD). In fact, we measure the maturation and acceptance of these approaches in terms of the degree to which their patterns and best practices are published in the literature. There have been some initial efforts to capture MDSD patterns associated with process and organization, domain modeling, tool architecture, and application platform development. We expect that the influence of patterns and pattern languages on aspect-oriented software development will be similar to that on model-driven software development.

- *Distributed real-time and embedded systems*. Developing high-quality distributed real-time and embedded systems is hard and remains somewhat of a ‘black art.’ A forthcoming book [16] documents patterns for developing distributed real-time and embedded software, based on material from pattern writing workshops at the OOPSLA and PLoP conferences. We expect the body of patterns for distributed real-time and embedded systems to continue to grow in the future since progress in this domain is essential for developing viable mission- and safety-critical applications.

- *Quality of service (QoS) for commercial-off-the-shelf (COTS)-based distributed systems*. To reduce development cycle-time and cost, such distributed systems are increasingly being developed using multiple layers of COTS hardware, operating systems, and middleware components. It is hard, however, to configure COTS-based systems that can simultaneously satisfy *multiple* QoS properties, such as security, timeliness, *and* fault tolerance. As developers and integrators continue to master the complexities of providing end-to-end QoS guarantees, we expect to see an increase in documented patterns that help others configure, monitor, and control COTS-based distributed systems that possess a range of interdependent QoS properties.

- *Mobile systems*. Wireless networks have become pervasive and embedded computing devices are become ever smaller, lighter, and more capable. Likewise, Internet services, ranging from Web browsing to on-line banking, are now accessed from mobile systems. Mobile systems present many challenges, however, such as managing low and variable bandwidth and power, adapting to frequent disruptions in connectivity and service quality, diverging protocols, and maintaining cache consistency across disconnected network nodes. We expect that experienced mobile systems developers will document their expertise in pattern form to help meet the growing demand for best software development practices in this area.

- *Software architecture*. Despite an increase in the number of documented pattern languages, the software industry has no parallel to the comprehensive handbooks found in other design disciplines. Although the existing patterns literature has made steady progress towards creating handbooks for software engineers the final goal has not been reached. As mentioned earlier, Grady Booch has joined this effort and is collecting thousands of patterns to create a *Handbook of Software Architecture* that exposes their essential roles and relationships and permits comparisons across domains and architectural styles.

- *Group interaction*. Although many patterns for human-computer-human interaction have been published in the recent years, no integrated view of these patterns is yet available. Moreover, not all areas of group interaction in an electronic environment are yet covered by patterns—for example, virtual worlds and games that support massive numbers of players. Given the growing demand for electronic collaboration support, we therefore predict that more patterns and pattern languages will be published in the area of human-computer-human interaction in the near future.

- *Web 2.0*. The Web increasingly provides the context for more dynamic and open business models, where 'harnessing collective intelligence' becomes the means of business and next-generation Web (known as 'Web 2.0') becomes the medium. The initial set of Web 2.0 patterns documented by Tim O'Reilly (*www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html*) captures some key emerging and proven practices in this space. We predict that more patterns will be documented as experience in this domain grows.

- *Business transaction and e-commerce systems*. Many business information systems, such as accounting, payroll, inventory, and billing systems, are based on transactions. The rules for processing transactions are complex and must be flexible to reflect new business practices and mergers. Business systems must also handle increasingly large volumes of transactions on-line. The meteoric growth of e-commerce on the Web has exposed many business-to-business systems directly to consumers. Despite the importance of these systems, relatively little has been written about their robust and secure analysis, architecture, or patterns. We expect the body of patterns on transactions and e-commerce to grow.

- *Process and organizational structure*. The growing adoption of agile development processes and experience with these processes suggests we will continue to see a growth in corresponding pattern literature. Some of this literature will focus on macro-process aspects, such as overall lifecycle and interaction with the business, and some will focus on micro-process aspects, such as test-driven development, refactoring, and tool usage.

- *The Gang-of-Four*. Many books and papers have been published about the Gang-of-Four patterns in the past, with no end in sight. An entire secondary industry will continue to feed the unquenchable demand for ruminations about the Gang-of-Four work. Examples include companions on how to implement the patterns in other programming languages, essays on missing ingredients, alternative solutions, and the proper scope of the patterns, and discussions about whether one or other pattern is not a pattern, or is outdated. There is talk of a second edition of the Gang-of-Four book, although the time frame for its release has not been finalized.

- *Pattern theory*. Work on the underlying theory of the pattern concept will continue over the coming years, focusing on deepening the knowledge about known facets of the pattern concept, such as pattern sequences, and exploring new and most recent views onto it, such as "problem frames," which name, bound, and describe recurring types of problems. Although there is still much scope for consolidation, clarification, and communication of theoretical concepts, we believe that the most interesting trends in future are associated with the domains in which patterns are documented, rather than in theory surrounding patterns.

Patterns and pattern languages will also be published for areas other than those listed above. We believe, however, that these topics are the most promising based on our current knowledge of work in the field. Naturally, our predictions of the future of patterns above include a healthy dose of uncertainty and are based upon our knowledge of the pattern community, its ongoing and planned interests and research activities, insofar as we are aware of them, and the future directions we are personally involved with or that we consider most fruitful.

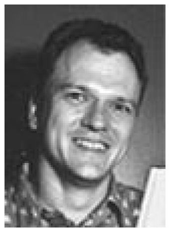
References

1. E. Gamma, et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addision Wesley, 2004.
2. F. Buschmann, et al., *Pattern-Oriented Software Architecture, Volume 4: A Pattern Language for Distributed Computing*, Wiley, 2007.
3. F. Buschman et al., *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns* Wiley, 1996.
4. G. Utas, *Robust Communications Software: Extreme Availability, Reliability and Scalability for Carrier-Grade Systems*, Wiley 2005.
5. M. Schumacher et al., *Security Patterns: Integrating Security and Systems Engineering*, Wiley 2006.
6. C. Steel et al., *Core Security Patterns: Best Practices and Strategies for J2EE Web Services, and Identity Management*, Prentice Hall, 2005.
7. M. Pont, *Patterns for Time-Triggered Embedded Systems*, Addison-Wesley 2001.
8. J. Noble et al., *Small Memory Software, Patterns for Systems with Limited Memory*, Addison-Wesley, 2000.
9. E. Evans, *Domain-Driven Design*, Addison-Wesley 2003.
10. G. Meszaros, *XUnit Test Patterns: Refactoring Test Code*, Addison-Wesley, 2007.
11. J. Coplien et al., *Organizational Patterns of Agile Software Development*, Prentice Hall 2004.
12. D. Manolescu et al., *Pattern Languages of Program Design: Volume 5*, Addison-Wesley, 2006.
13. F. Buschmann, et al.,*Pattern-Oriented Software Architecture, Volume 5: On Patterns and Pattern Languages*, Wiley 2007.
14. M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley 2002.
15. *G. Hohpe, et al., Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley 2003.*
16. L. Dipippo et al, *Design Patterns for Distributed Real-Time Embedded Systems*, Springer 2007.

Author Bios

Frank Buschmann is a Principal Engineer at Siemens Corporate Technology in Munich, Germany. His research interests include Object Technology, Software Platforms and Product Lines, Application Frameworks, and specifically Patterns. In his development work, Frank has lead the design and implementation of several large-scale industrial software projects, including business information, industrial automation, and telecommunication systems. Frank has published widely in areas like software architecture, software reuse, reflection, application frameworks, and patterns -- most visible in his co-authorship of four volumes in the "Pattern-Oriented Software Architecture" (POSA) series of books. Frank also served as a member of the ANSI C++ standardization committee X3J16 from 1992 to 1996.

Siemens AG
Corporate Technology
Software and Engineering
CT SE 2
Otto-Hahn Ring 6
81739 Munich / Germany
Phone: +49-89-636-49323
Fax: +49-89-636-45450
mailto: Frank.Buschmann@siemens.com



Kevlin Henney is an independent consultant and trainer based in the UK. His interests and areas of expertise include programming languages, object orientation, patterns, software architecture, and development process. His clients range from small start-ups to global corporations, across a variety domains, including telecommunications, information services, logistics, and finance. Kevlin has published over 300 papers, articles and columns, and is a coauthor of two volumes in the "Pattern-Oriented Software Architecture" (POSA) series of books. He has been a member of committees and boards for conferences, publications and other endeavors, including the BSI and ISO C++ standardization committees.

Kevlin Henney
kevlin@curbralan.com
Curbralan Ltd
17 Tyne Road
Bishopston
Bristol
BS7 8EE
United Kingdom
Tel: +44 117 942 2990
Fax: +44 870 052 2289

Dr. Douglas C. Schmidt is a Professor of Computer Science and Associate Chair of the Computer Science and Engineering program at Vanderbilt University. He also serves as the Chief Technology Officer for PrismTech and is a visiting scientist at the Software Engineering Institute. He has published 9 books and over 350 technical papers that cover a range of research topics, including patterns, optimization techniques, and empirical analyses of software frameworks and domain-specific modeling environments that facilitate the development of distributed real-time and embedded (DRE) middleware and applications running over high-speed networks and embedded system interconnects. He received his Ph.D. in Computer Science at University of California, Irvine and is a member of IEEE.

Douglas C. schmidt
d.schmidt@vanderbilt.edu
Institute for Software Integrated Systems
2015 Terrace Place
Vanderbilt University
Nashville, TN 37064
(TEL) 615 294 9573
(FAX) 615 343 7440