Editor: **Michiel van Genuchten**
MTOnyx
genuchten@ieee.org

Editor: **Les Hatton**
Kingston University
l.hatton@kingston.ac.uk

Twenty Years of Patterns' Impact

Gregor Hohpe, Rebecca Wirfs-Brock, Joseph W. Yoder, and Olaf Zimmermann

This column celebrates the 20th year of software patterns. *IEEE Software* advisory board members teamed up with members of the Hillside Group, a nonprofit organization that promotes the use of patterns and pattern languages, to reflect on the state of the practice and impact of patterns. —*Michiel van Genuchten and Les Hatton*

GOOD ADVICE IN software design is difficult to come by. General design principles can guide us, but reality tends to force trade-offs between seemingly conflicting goals, such as flexibility and maintainability against size and complexity. Likewise, code libraries can go a long way in helping us avoid reinventing the wheel, but the vision of lesser-skilled developers effortlessly wiring together ready-made components remains fiction.

Design patterns have helped narrow this gap by documenting a well-working solution to a problem that occurs repeatedly in a given context. Instead of presenting a copy-and-paste-ready code snippet, patterns discuss forces impacting the solution design. Examples of such forces are performance and security in Web applications: encryption and decryption algorithms improve security but introduce processing overhead. Ward Cunningham once described the best patterns as your older brother teaching you how to do something right.[1]

Although patterns have become popular, their impact as a design technique is more difficult to quantify than the impact of a specific software product (which is what previous installments of this column have examined). This installment highlights both the breadth of patterns available after 20 years of pattern-writing conferences and the depth of impact some patterns have had on open source software.

continued on p. 84

0740-7459/13/$31.00 © 2013 IEEE

continued from p. 88

How It All Began

Building architect and philosopher Christopher Alexander inspired Kent Beck and Ward Cunningham to write their first small pattern language in 1987 for designing Smalltalk windows. In 1993, Beck and Grady Booch sponsored a mountain retreat in Colorado that triggered the formation of the nonprofit Hillside Group to foster pattern writing through the Pattern Languages of Programming (PLoP) conference series, which is celebrating its 20th successful year. PLoP conferences follow a highly collaborative style based on "shepherding" before submission and peer-based feedback workshops during the conference. Many successful pattern papers and books have emerged from this process.

In 1994, Erich Gamma and his colleagues' *Design Patterns* catapulted the concept of patterns to a broad audience; as of this writing, it has sold more than 500,000 copies in 13 languages.[2] Two years later, Frank Buschmann and his colleagues produced the first volume of the *Pattern-Oriented Software Architecture* series,[3] closely followed by Martin Fowler's *Analysis Patterns*.[4] (Resources for further reading are available elsewhere.[5–9]) The pattern format's apparent success even tempted some authors and publishers to gratuitously add the word "patterns" to their titles—the price of success, we feel. As of 2013, an Amazon search on "patterns" among computer and technology books yields more than 5,500 unique hits (including a minor number of false positives on visual pattern detection).

The early hype around patterns has settled, and people realize that patterns neither replace design skills nor solve all problems. Still, well-crafted patterns provide valuable nuggets of relevant advice based on actual experience. Because learning by doing (learning from making mistakes) often isn't an option for real-world projects, patterns can provide a way to learn from others' experience (and mistakes, which can make good antipatterns).

No Sign of Pattern Fatigue

The widespread diversity of pattern domains makes determining the exact number of documented patterns difficult. Linda Rising's *The Pattern Almanac 2000* listed more than 1,000 patterns.[10] The PLoP conferences, sponsored by the Hillside Group (www.hillside.net), have accepted more than 1,500 papers. The submission rate to those conferences has been constant, at approximately 100 papers per year. A conservative estimate of four patterns per paper, plus all the books and online catalogs, puts today's number of published patterns at more than 7,500, and growing. Patterns cover a range of computing topics, including cloud computing, distributed systems, concurrent and parallel programming, user interface design, mobile app development, adaptive systems, sustainable architectures, domain-specific patterns, metaarchitectures, workflow, fault-tolerant systems, and security.

> A conservative estimate puts today's number of published patterns at more than 7,500, and growing.

Many people accept the definition of a pattern as a proven solution to a problem in a context. In *The Timeless Way of Building*, Christopher Alexander clarifies that "the pattern is, in short, at the same time a thing, which happens in the world, and the rule which tells us how to create that thing, and when we must create it."[11] Patterns present a reusable solution, provide information about its usefulness and trade-offs, and encapsulate knowledge about proven best practices.

For example, many integration architectures include the Broker pattern, which acts as an intermediary between clients and servers and handles message routing, including serialization and deserialization of message content.[2] The Web's communication infrastructure implements this pattern; workflow engines such as YAWL (Yet Another Workflow Language) also include rich implementations of this pattern.[12]

Many patterns are part of a pattern library; examples include http://developer.yahoo.com/patterns and www.securitypatterns.org. Many companies, including Amazon, Google, IBM, Lucent, Microsoft, Oracle, and Siemens, have written similar pattern collections, some of which are available in books and on websites. One example of such pattern collection is the IBM patterns for e-business catalog. Among many other recurring designs, it featured implementations of the Enterprise Service Bus in the context of IBM WebSphere products.[13] Connected sets of interrelated patterns building on each other can form *pattern languages*, which support a generative, domain-specific development process.[14] There's even a pattern language for writing patterns.[15]

Enterprise Integration Patterns

Patterns' success in software architecture and design has motivated attempts to integrate them more closely into programming tools to boost productivity and more closely align design and implementation mindsets. Alas, most attempts have stumbled because patterns are inherently a medium for documenting and passing knowledge between humans, not a programming construct. Still, some pattern languages have directly affected how software solutions are built.

Around 2003, the term *Enterprise Service Bus* (ESB) gained traction for describing the integration platform for service-oriented architectures. ESBs route, filter, and transform XML messages between services; they represent the evolution of traditional enterprise application integration products that implement the Broker pattern. Ironically, although these products aimed to unify the Tower of Babel of disparate enterprise applications, no shared vocabulary to describe such solutions' design was available.

Developers of open source ESB implementations that aimed to overcome this apparent gap soon realized that Enterprise Integration Patterns (EIPs) provide a coherent vocabulary of 65 patterns,[7] ranging from integration styles to message routing and transformation. This can describe a large portion of meaningful ESB solutions. In the absence of an ESB industry standard, the open source projects adopted the EIP vocabulary as a de facto standard.

Open Source ESBs

Since the emergence of open source ESBs in 2005, almost a dozen open source ESB products have embedded the EIP language in their products' domain-specific languages or programming models. The most widespread examples are Mule (www.mulesoft.org), Apache Camel (http://camel.apache.org), WSO2 ESB (http://wso2.com/products/enterprise-service-bus), Spring Integration (http://projects.spring.io/spring-integration), and OpenESB (www.open-esb.net).

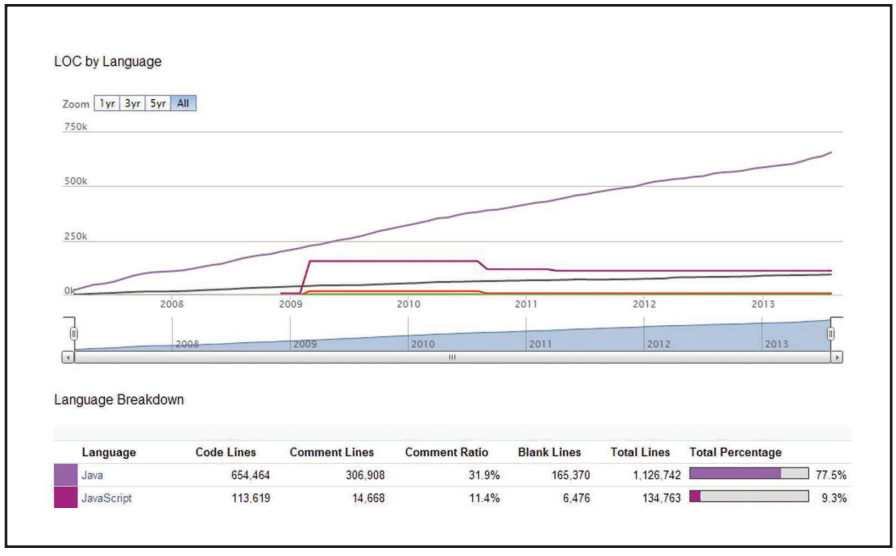

FIGURE 1. Apache Camel core code growth over time. Linear growth of the Java code base suggests a stable committer community and sustained engagement. The amount of JavaScript jumped in 2009, but was later reduced, likely to the availability of libraries and frameworks.

The nature of open source projects makes tracking code size relatively easy. However, tracking volume is relatively difficult because sales figures don't exist and download numbers are often tainted by mirroring, caching, or automated downloads.[13] Apache Camel comprises some 890 KLOC, created by 62 committers over the course of more than 18,000 individual commits over six years. The (Java) code base's growth has been amazingly linear (see Figure 1), which suggests consistent engagement by a stable set of committers. Commercial adaptations of the open source core—for example, by Red Hat or Talend—augment the code base significantly with design or runtime management tools.

Download figures derived from Maven Central have averaged about 25,000 a month with a peak of more than 30,000 in July 2013—higher than YAWL, which reported about 1,000 downloads per month in 2010.[13] Mule reports 3.6 million downloads on its homepage but doesn't indicate whether all of them are individual user-initiated downloads.

Community engagement provides another insightful metric of open source success. Apache Camel community traffic quickly ramped up after its initial release in 2007 and holds steady at about 2,500 messages a month. This indicates a healthy community that collaborates to resolve issues and drives the product's evolution. For comparison, Mule's community page counts more than 150,000 members, and its forum counts 26,600 total posts.

Patterns as a Design Tool

After the EIP vocabulary's integration into those products proved popular, some ESB projects went one step further and adopted the EIP pattern sketches as the visual language for their design studios. For example, developers can access the EIP icon language within the Red Hat Fuse IDE

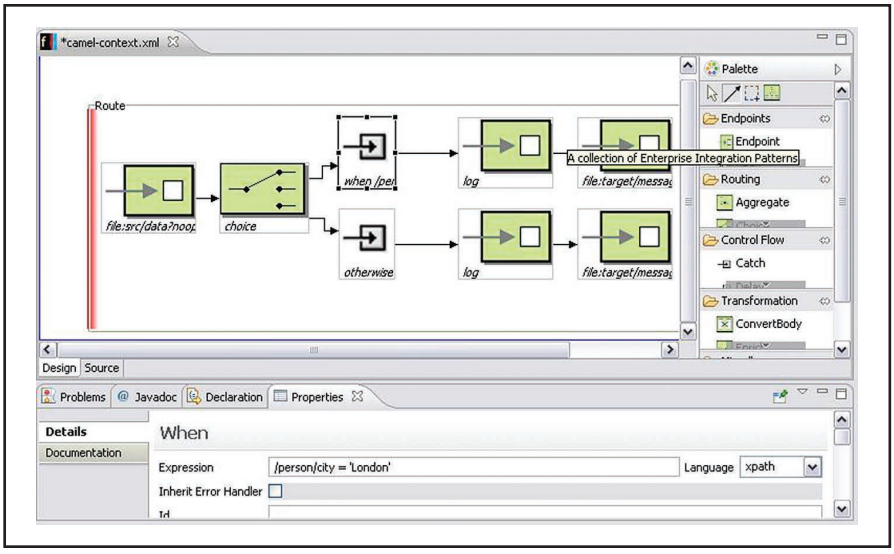

FIGURE 2. Creating messaging solutions using the visual pattern language from Enterprise Integration Patterns (EIPs)[7] inside the Redhat Fuse IDE (integrated development environment). Messages arriving from a file-based message endpoint are routed by a content-based router to one of two potential message endpoints based on the city specified inside the message content. The content-based router pattern describes a reusable design for routing messages to a correct recipient based on message content.

FIGURE 3. Playing cards based on Enterprise Integration Patterns. The visual pattern language allows for an interactive, almost playful usage of the patterns. Each card displays the pattern icon together with the name and solution statement.

(integrated development environment) or Mule Studio. Unlike prior, somewhat contrived "visual programming" attempts, the simple pipes-and-filters architectural style of asynchronous messaging solutions makes this visual composition of patterns natural. Figure 2 shows a visual Camel route that directs an incoming message to one of two possible message endpoints via a message router. ESB developers can now think, design, communicate, and implement their solutions using the EIP vocabulary, even if they're using different runtime platforms.

The EIP playing-card deck handed out at the inaugural CamelOne conference is likely the most creative pattern adaptation to date (see Figure 3). Each card shows a pattern from the pattern language together with the solution statement. It's satisfying to see design patterns, which were created to improve human communication and collaboration, finding their way (literally) into the hands of architects and engineers in such an approachable, useful way.

The statistics we presented here indicate that pattern languages have had a broad impact on the software design community over the past 20 years. Many research questions around patterns remain

open, however. For example, good patterns aren't always easy to find, which invites more work to organize and catalog the large body of existing patterns. We also envision pattern language authoring tools, perhaps using semantic wiki technologies. Finally, pattern-centric design tools promise to be more appealing to the software engineers than mere component-and-connector drawing tools.

Will the patterns community ever lose momentum? We don't think so: existing pattern languages will continue to be implemented as domain-specific languages, just like EIPs. And domains still exist for which patterns have yet to be captured. For instance, typical conversations both between applications (via technical protocols) and between humans (such as via social networks) could be preserved in pattern form.

The future of patterns is bright. We invite you to help shape it by furthering the development of a pattern tool or writing and sharing your design wisdom in pattern form.

References

1. W. Cunningham, "Tips for Editing Patterns," Dec. 2002; http://c2.com/doc/TipsForEditors.html.
2. E. Gamma et al., *Design Patterns*, Addison-Wesley Professional, 1994.
3. F. Buschmann et al., *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*, John Wiley & Sons, 1996.
4. M. Fowler, *Analysis Patterns: Reusable Object Models*, Addison-Wesley Professional, 1996.
5. J. Kerievsky, *Refactoring to Patterns*, Addison-Wesley Professional, 2004.
6. M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley Professional, 2002.
7. G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional, 2004.
8. E. Evans, *Domain Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley Professional, 2003.
9. V. Vernon, *Implementing Domain-Driven Design*, Addison-Wesley Professional, 2013.
10. L. Rising, *The Pattern Almanac 2000*, Addison-Wesley, 2000.
11. C. Alexander, *The Timeless Way of Building*, Oxford Univ. Press, 1979.
12. M. Adams, A.H.M. ter Hofstede, and M. La Rosa, "Open Source Software for Workflow Management: The Case of YAWL," *IEEE Software*, vol. 28, no. 3, 2011, pp. 16–19.
13. M. Keen et al, *Patterns: Implementing an SOA Using an Enterprise Service Bus*, IBM, 2004; www.redbooks.ibm.com/abstracts/sg246346.html.
14. F. Buschmann, K. Henney, and D. Schmidt, "Past, Present, and Future Trends in Software Patterns," *IEEE Software*, vol. 24, no. 4, 2007, pp. 31–37.
15. G. Meszaros and J. Doble, *A Pattern Language for Pattern Writing*, Hillside Group; http://hillside.net/index.php/a-pattern-language-for-pattern-writing.

GREGOR HOHPE is chief enterprise architect at Allianz SE and a member of the Hillside Group. Contact him at info@enterpriseintegrationpatterns.com.

REBECCA WIRFS-BROCK is president of Wirfs-Brock Associates and treasurer of the Hillside Group. Contact her at rebecca@wirfs-brock.com.

JOSEPH W. YODER is president of The Refactory, Inc., and of the Hillside Group. Contact him at joe@refactory.com.

OLAF ZIMMERMANN is a professor and institute partner at the Institute for Software at the University of Applied Sciences of Eastern Switzerland, Rapperswil (HSR FHO). Contact him at ozimmerm@hsr.ch.

IEEE Software (ISSN 0740-7459) is published bimonthly by the IEEE Computer Society. IEEE headquarters: Three Park Ave., 17th Floor, New York, NY 10016-5997. IEEE Computer Society Publications Office: 10662 Los Vaqueros Cir., Los Alamitos, CA 90720; +1 714 821 8380; fax +1 714 821 4010. IEEE Computer Society headquarters: 2001 L St., Ste. 700, Washington, DC 20036. Subscription rates: IEEE Computer Society members get the lowest rate of US$56 per year, which includes printed issues plus online access to all issues published since 1984. Go to www.computer.org/subscribe to order and for more information on other subscription prices. Back issues: $20 for members, $209.17 for nonmembers (plus shipping and handling).

Postmaster: Send undelivered copies and address changes to *IEEE Software*, Membership Processing Dept., IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854-4141. Periodicals Postage Paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Agreement Number 40013885. Return undeliverable Canadian addresses to PO Box 122, Niagara Falls, ON L2E 6S8, Canada. Printed in the USA.