UDC 681.3.06

The Culture of Patterns

James O. Coplien

Vloebergh Professor of Computer Science, Vrije Universiteit Brussel, PO Box 4557, Wheaton, IL 60189-4557 USA
JOCoplien@cs.com

Abstract. The pattern community came about from a consciously crafted culture, a culture that has persisted, grown, and arguably thrived for a decade. The culture was built on a small number of explicit principles. The culture became embodied in its activities—conferences called PLoPs that centered on a social activity for reviewing technical works—and in a body of literature that has wielded broad influence on software design. Embedded within the larger culture of software development, the pattern culture has enjoyed broad influence on software development worldwide. The culture hasn't been without its problems: conflict with academic culture, accusations of cultism, and compromises with other cultures. However, its culturally rich principles still live on both in the original organs of the pattern community and in the activities of many other software communities worldwide.

1. Introduction

One doesn't read many papers on culture in the software literature. You might ask why anyone in software would think that culture is important enough that an article about culture would appear in such a journal, and you might even ask yourself: just what *is* culture, anyhow?

The software pattern community has long taken culture as a primary concern and focus. Astute observers of the pattern community note a cultural tone to the conferences and literature of the discipline, but probably view it as a distant and puzzling phenomenon. Casual users of the pattern form may not even be aware of the cultural focus or, if they are, may discount it as a distraction. However, the bulk of the pattern community embraces a common culture that guides its activities to socialize and publish a design literature. These cultural underpinnings have value to the pattern community in achieving its goals. More broadly, these cultural foundations have grown to support technological progress, exchange of ideas, professional activities, and the overall cultural tone in the broader computing community. The practices and tenets of the software pattern community have value to software development.

Just what is the "software pattern community?' (Hereafter, in this paper I'll simply use the term "pattern community" to mean "software pattern community.") Here, I define the pattern community as those people who work in community to write and refine patterns and pattern languages. We can regard the pattern community as including those who embrace the pattern culture. Patterns also touch many people outside the pattern community. There is a difference between being in a culture and being influenced by a culture. French culture might include an appreciation of wine, bread and cheese as the foundation of everyday nourishment, but to drink French wine does not make one part of the French culture.

Each software development group, or company, has its own culture. Many software development cultures cut across organizational boundaries in the sense that their constituents adopt a common set of values, behaviors, vocabulary, history, and stories. We can define a Smalltalk culture in this sense, or a Macintosh or Linux culture. Not all technologies or fads define cultures; it is difficult to recognize a "C++ culture" in any normative sense of the word. The pattern culture is like the Smalltalk, Macintosh or Linux cultures in that it is normative: we can identify common traits of its constituents. Its community is broader than that of any single organization or social sector: it crosses academic, industrial, national, ethnic, gender, and natural language boundaries.

For those who like dictionary definitions, consider the following from the first definition of culture (Dictionary):

The totality of socially transmitted behavior patterns, arts, beliefs, institutions, and all other products of human work and thought.

These patterns, traits, and products considered as the expression of a particular period, class, community, or population: *Edwardian culture; Japanese culture; the culture of poverty.*

These patterns, traits, and products considered with respect to a particular category, such as a field, subject, or mode of expression: *religious culture in the Middle Ages; musical culture; oral culture.*

The predominating attitudes and behavior that characterize the functioning of a group or organization.

I also like Luke Hohmann's selection of elements that make up culture, which he uses to assess software development (Hohmann 1996). I will use these elements to focus the considerations of the rest of the paper:

- Language
- Normative behavior
- Values
- Symbols
- Stories
- Rituals

Patterns feature all of these elements in their articulation of the recurring structures of relationships and activity that solve social and cultural problems. Patterns are about capturing and defining a domain vocabulary. They capture the normative practices of design in software. They reflect a value system. One can take a semiotic view of patterns as the deeper signs behind the language of design. Patterns are based on stories and design rituals. The pattern community thrives on its own rituals and, to a lesser degree, on the stories that give it an identity.

Proper culture emerges over time through "socially transmitted behavior[s]". The pattern culture came about in part from such evolution. What is noteworthy is that it was originally a consciously crafted culture. In this respect, it is different from the Smalltalk, Linux or Macintosh cultures. This paper in large part focuses on that crafting activity and reflects on its degree of success.

Technically, according to Kroeber (1948, pp. 82—88) the pattern culture is a "semi-culture" or a partial culture embedded in a larger culture, that of software development. One may even argue that it is a society rather than a culture; however, the two are difficult to separate in human organizations (Kroeber, pp. 7–10). The relationship between the software pattern culture and other pattern cultures that take inspiration from Alexander's work, including the architectural pattern culture in particular, are complex but operationally are quite weak. These perspectives do not change the analyses or arguments about the properties of the pattern culture, however, and in this paper I adopt a cultural posture.

A culture's history can speak volumes, and this paper will employ history heavily. History is always written from a perspective and is open to interpretation, and articulation neither of history nor of culture constitutes truth. Some of the people who were there and who were directly involved brought less of a cultural agenda and perspective to the table than others did, and they have reported in their feedback to me that they don't remember as conscious a cultural agenda as I relate here. Nevertheless, all but one of them carried an agenda for change: whether they were thinking of such change in cultural terms is less important to me than that they in fact affected cultural change. I have done my best to socialize this paper with colleagues as varied as Linda Rising (a frequent spokesperson for the software pattern community), Frank Buschmann (an early and active member of the pattern community) and David Weeks (an Australian building architect who practices patterns and who is also a friend of the software pattern community) to help round out the perspective.

2. The Culture as it was Designed

Some of the people who would later found the pattern community first came together in May of 1993 at a workshop at IBM in Thornwood, New York. That group shared an interest in building a body of software design literature. In August of 1993, a group of seven people met in Colorado to evaluate how patterns might redress the problems of the object community. Those seven reflected strong leadership roles in the object community: Ken Auer, Grady Booch, Ralph Johnson, Hal Hildebrand, Kent Beck, Ward Cunningham, and Jim Coplien. Many of the cultural norms would come together at that meeting. The group called itself the Hillside Group, after a group design exercise they experienced together on a hillside at this Colorado meeting, in the shadow of Buffalo Mountain.

This group of programmers who individually had been influential in software design came together in 1993 under the common belief that the object oriented software community was suffering from serious problems. They felt that object orientation had failed to live up to its "promises" of reuse and productivity. These promises were rooted in a worldview that held that an object partitioning of the world captured the structure of the world. Object orientation was suffering from myopia: while designers managed individual objects and classes, they failed to grasp system concerns. Those concerns could be found in the relationships between objects, and object orientation had failed to embrace this notion of relationships. The Hillsiders believed that the prevailing software development culture of the 1980s had been handicapped by this myopia.

(David Weeks notes that "this story in itself constitutes an 'origin myth' for the software pattern community. Such origin myths are 'crafted' in order to reinforce or perpetuate a certain culture.")

Cultural practices exist to solve problems. It was at this first meeting that the problem and solution were identified. The key problem was that modern computing literature failed to convey the key low-level elements of design that contributed heavily to the success of a project. Those insights reflected sometimes tacit knowledge that lived in the heads of everyday programmers. Because everyday programmers tend not to publish, and because the knowledge was tacit, this knowledge had gone untapped in the computing culture of the 1970s and 1980s that valued publications from industry cult figures. The pattern community envisioned a new body of literature, written by practicing programmers, to elicit and capture these key design notions. However, the goal went beyond encouraging programmers to write and publish, because such encouragement would likely not be enough: Such "folk" literature was not valued by the existing publication processes and venues. We needed a *body* of literature whose content reflected the values that had brought the Hillsiders together. We had seen success with isolated examples of such literature: the Design Patterns that had been informally circulating in object-oriented circles for the preceding two or three years; the C++ Idioms book (Coplien 1992), and some Smalltalk Best Practices had both enjoyed some influence. To define a body of literature rooted in a value system required a cultural framework. It was out of such considerations that the pattern culture was born.

It would be at the next Hillside meeting at Ben Lomond, California, in April 1994 that the founders of the software pattern discipline would outline the pattern culture. That meeting brought together the original seven, less Hal Hildebrand, and with the addition of Bruce Anderson, Desmond D'Souza, Richard Gabriel, Richard Helm, Norm Kerth, and John Vlissides.

To a large degree, this group was conscious about defining a culture (or at least making a dent in the prevailing culture of the day). Some of them had a sense of their place in history. Kent Beck's invitation to the Colorado event, dated 15 July 1993, mused: "Please get ready for a happening, ripples of which will be felt for many years to come." On the other hand, their ideas had not achieved the stature of a master plan. The group had doubts about how the anticipated activities would fit together, and about logistics in general. Richard Gabriel remarked, "Well, let's just go and pretend like we know what we're doing, and everything will be fine."

Here, I summarize six main tenets of the culture that the early community laid down as the foundations of the pattern discipline. (There is nothing magical about the number "six," and these categories overlap somewhat.)

2.1. Valuing Specialization and Experience

The first and perhaps most fundamental principle of the pattern community is to value the expertise that comes from domain-specific experience. In the past, software had valued novelty: object orientation was good not because it had proven itself in terms of reuse or productivity, but because it was new. Because the status quo before objects had failed to meet the unrealistic industry expectations for reuse and productivity, it was presumed to be inadequate. Further, any technology that even mentioned reuse and productivity was presumed to be superior to the previous techniques. Object orientation gained such a reputation, ostensibly through such presumptions. The pattern community—which comprised major leaders in the object oriented discipline of that day—had come to the point of realizing that object orientation had also failed on its promises. The community, tired of promises, decided to retreat to more concrete foundations: real architectures and code that experience had proven to work. Such proof was embodied in the expertise of "common" programmers who faced these challenges every day.

Part and parcel to this value was the trust that the community placed in its authors. Authors were encouraged to write about areas in which they were expert, and the community trusted authors to do so. This was not meant to be an exclusive policy; the value system held that every programmer has insights worth committing to writing (even though not every programmer actually does). The goal at every PLoP conference is to support authors so that as many papers as possible would be admitted to the conference. (PLoP stands for Pattern Language of Programs: the conferences focus more on code and everyday software development, than on artifacts from high stations of architecture or management.) The conference provided *shepherds*: experienced pattern writers who were paired with conference authors to help them improve their papers. The Hillsiders formed the initial shepherding group, but the group grew quickly as the community matured.

One corollary to this value was, as Brian Foote eloquently expressed, "an aggressive disregard for originality." Original ideas (and particularly those that emerged from non-practicing disciplines such as academics) by definition had no record of accomplishment. They might benefit from grounding in a formal proof, but correct proofs of complex constructs and algorithms seemed elusive for the most important considerations of complex system design. Instead of originality, the community embraced experience: experience that was embodied in true expertise. Expertise would come to be distinguishable from, and distinguished from, opinion. Most methodologies, design practices, and software fads were driven not by grounded experience but by hopes placed in the opinions of key spokespersons in the software cultures of the time. This was contrary to how patterns would later come to be viewed, because patterns in fact represented a strong return to scientific empiricism that could be contrasted with the ad hoc reasoning that had guided much of software until that time.

Closely related to this value was the value of inclusiveness. The pattern community sought engagement across disciplines. In spite of its philosophical differences with academia as an institution, it didn't close the door on academics that wanted to become involved. Several died-in-the-wool academics did find their way into the community: Ralph Johnson and Oscar Nierstrasz are two good examples. The community was careful to not impose the object orthodoxies of its roots on itself.

Other principles of the community were closely tied to this one. Methodologists were promising universal methods that solved any problem, independent of domain expertise; the patterns folks wanted to stay grounded in everyday experience. This value is strengthened through Writers' Workshops and shepherding, both done in community. The value is weakened by people who simply publish works in pattern form, a recent trend that devalues the community.

2.2. Dignity for Programmers

A second principle of the pattern community, closely related to the first, was that of dignity for programmers: valuing the contributions of everyday people as much as those of methodologists and high academics. Christopher Alexander's work on architecture provided the inspiration for this vision. His own work cast aspersions on architects as insecure leaders interested more in their own survival and visibility than in the quality of life of their clients or of society as a whole.

The software pattern community transferred this same suspicion to methodologists. Instead of gathering and publishing ideas of broad scope or of deep principles traditionally of interest to theoreticians, the community embraced practical, day-to-day knowledge of interest and value to practicing programmers. This principle was borne out of the founders' experience that too many academic papers successfully passed the gauntlet of exclusive conference publication, and then were triumphantly presented at a single conference session only to finish out their existence as unread, dead literature that collected dust on the shelves of libraries and academic offices. In concert with the first principle of valuing expertise, the pattern community felt that the real expertise might lie in the heads of practitioners rather than coming from methodologists and "famous people."

Another facet of this value was the quality of work life for software developers. The Hillsiders felt that some American software development shops had become the twentieth century equivalent of the sweatshops of the industrial revolution: long hours, hard work, in environments where programmers were viewed like machines that took specifications in at one end and produced code at the other. The Hillsiders wanted to recognize developers for their intellectual contribution to the end product and accord them the dignity due their station.

While the pattern community founders were attentive to the needs of the customers that *use* software, much of the energy focused on the needs of the developers who *produce* software. This distinguishes the software pattern community from the architectural pattern community. The architectural pattern community elevates the experience of the building *users*, whereas the software pattern community elevates the experience of the *craftspersons* and *technicians*.

2.3. Supplanting Academic Tradition

A broad third tenet of the culture is to replace academic traditions, which were viewed as outdated in their service to the industry, with new traditions. The Hillside Group introduced explicit cultural rituals, activities, and guidelines to exemplify this principle and make it manifest to both cultural participants and observers. These provisions fall into three major areas: fun, expertise inversion, and empiricism.

The planning of the original pattern conferences anticipated games as a major component. Having fun activities such as games provided three cultural foundations. The first was to serve as an outward sign of distancing from the academic community, which the pattern community viewed as being too serious about itself. The second was to provide an activity that balanced that staid and sometimes tedious effort of the critical review that takes place in Writers' Workshops. A third goal was to fuel lateral thinking for its benefits to creative processes—such as writing. A standing conference in the UK, which would come to be known as OT or more affectionately "The Circus," and which had taken place in Oxford and Cambridge, provided the inspiration. George Platts had organized the games at these conferences, and the PLoP conferences enlisted his support and help in bringing the game culture to the new conferences. He came to be known as the "querdankenmeister" of the community—a clear cultural role.

Writers' Workshops would in fact become a staple not only of pattern conferences but also of the pattern community in general, and are probably the strongest normative behavior and ritual of the pattern community. In a Writers' Workshop, authors come together to constructively critique each others' work. The review focuses both on positive aspects of the work, and on opportunities for improvement. The author of the work under review is silent for most of the review and is allowed to ask only questions of clarification at the end of the review. For more on writers' workshops, see (Gabriel 2002).

Expertise inversion means that instead of putting the decisions of topic selection, technical publication in the hands of organizers who control the fates of hopeful authors, the community instead defined structures and processes to support authors in successful publication. One important goal of the culture was make tacit knowledge explicit—especially that tacit knowledge kept in the heads of experts who had grass-roots experience. Some of the early pattern community members, such as Ward Cunningham, were frustrated about not being able to publish small nuggets of everyday Smalltalk code that exhibited elegance and mastery of the language at mainstream conference venues. The pattern community provided a publication outlet for those ideas. In that regard, some founders of the pattern community believed that the purpose of the pattern community was to stop or slow the decline of Smalltalk.

The pattern community was conscious about falling into the same traps of academia. Computer people love to talk about philosophy and theory, and there was plenty of pattern philosophy and theory to discuss. The Hillsiders committed to limiting the community to "real stuff" in its publications. Going beyond discussion of concrete patterns was referred to as "going meta," and the culture adopted strong prohibitions against going meta. ("Computer scientists will go meta at the drop of a bit" became a community watchword.) We were to write patterns—not to write *about* patterns. It was great to discuss the more abstract theories at conferences, but our writing should build on the value of expertise and experience and stick to experience.

Another small element of expertise inversion was the "somebody should" principle. If we found ourselves saying, "Somebody should pull together these patterns from domain X," it became a stipulation to immediately concretize the "somebody," and the concretization most often fell on the person who informally proposed the notion. The idea was that one did not have to ask permission of anyone in the pattern community if one wanted to do something; any person was as expert as any other in developing new activities. This principle of course served to grow the pattern community without the bureaucratic constraints that can plague the large professional organizations at the core of some software communities. For the same reason, the pattern community decided to go it alone with the PLoP conferences instead of seeking sponsorship from bodies such as the ACM or IEEE. The outlook might be characterized as an overall disregard for established authority—whether institutional, professional, or academic. It was a counter-culture.

The culture also distinguished itself by empirically grounding its work. It is certainly true that empiricism finds a home among honorable academics. However, the culture of object-oriented programming had been moving into a more esoteric space: formalism with little empirical contact, understandable by and understood by only tiny fractions of the object community. OOPSLA papers became increasingly inaccessible to everyday programmers as object orientation became an increasingly esoteric discipline. Rather than relying on the formal foundations and traditions of academia, the pattern culture decided to get its hands dirty. Empiricism would rule the day. This principle, of course, is in concert with the focus on everyday programmers and the "aggressive disregard for originality" of the culture's foundations in experience. This value also echoed a common value of the architectural pattern community, which rails against "empty formalism" in favor of empiricism.

2.4. Systems Thinking

The fourth principle is systems thinking. At the time, two extremes were driving much thought and work in the object community: reusable objects, and object-oriented methodologies. The reuse community advocated the design of individual objects (actually, classes) that corresponded to real-world entities. These objects could be stored away and catalogued, to later be assembled at will into running systems. Experience had shown that it was difficult to get the object structure "right," and most objects needed customization to fit a given application. This was thinking too small.

On the methodology side, objects were individual, local units of design and programming. The rise of software methodologies in the 1980s had given rise to a "big bang" style of development that started with class and object diagrams: a complete master plan that constrained the system structure before developers had written the first line of code. Objects and classes were created and shaped from first principles of object-oriented design. Historic experience was relegated to those few objects that had been cast into reuse libraries; the rest of the structure arose from first principles and ad-hoc techniques. Without feedback, a system can too easily start its life headed in the wrong direction. Methodologists had us thinking about too much too early, and were overly driven by anticipation.

Alexander believes in a process of piecemeal growth and local adaptation. This perspective aligned well with that segment of the object-oriented programming community that embraced prototyping. Prototyping practices were strongly rooted in the Smalltalk community, drawing from the flexibility of their development environments. Alexander also insisted on using flexible building materials (his favorite construction materials were easily shaped chicken wire and sprayed concrete), and took a strong stand against pre-manufactured parts:

> *Design is often thought of as a process of synthesis, a process of putting together things, a process of combination.*
>
> According to this view, a whole is created by putting together parts. The parts come first: and the form of the whole comes second.
>
> *But it is impossible to form anything which has the character of nature by adding preformed parts.*
>
> When parts are modular and made before the whole, by definition then, they are identical, and it is impossible for every part to be unique, according to its position in the whole. (Alexander, 1979: p. 368)

Alexander believed in the importance of the Whole, but that a sense of the Whole should come from within ourselves and from experience. We should always be aware of the Whole on which we are working; but we should not project that Whole too far into the future.

2.5. Sharing and Giving

The fifth and last major principle is that of sharing and giving. The Open Source community had already demonstrated some successes by this time, but most of their work took place in the realm of programming. Some of the early pattern people were interested in raising the level of discourse to embrace design—again, in the spirit of Alexander's work. Like Alexander's work, it was still to be grounded in everyday experience and in real work. However, the artifacts would be ideas, approaches, and architectures, rather than actual systems, libraries, or classes. If people would share the common micro-architectures that every project rediscovers through tedious work and long investment, then we would raise the level of the entire industry.

This value was in fact a key value in bringing the Hillside Group together. In an early exercise at the Hillside retreat in 1993, the original seven Hillside people enacted an on-site "design" of a conference center called "The Center for Object-Oriented Programming." This hypothetical (at least in retrospect) facility would be a place where programmers would come together with people from the domains they served for a free exchange of ideas that addressed shared concerns. It doesn't serve a company to compete on the basis of designs that everyone else is re-inventing, so the entire community would be made more effective by sharing ideas and then moving on to more specialized areas where companies could distinguish themselves in the market and compete. Code reuse hadn't worked; maybe design reuse would. It was that realization that provided a key hope and focal point for the furtherance of the community.

The pattern community has evolved to a network of sharing, a network that transcends corporate boundaries. The community body of literature provides one forum for sharing. The Writers' Workshops that take place at every pattern conference provide another powerful opportunity for sharing openly and interactively. As David Weeks commented on an earlier draft of this paper, "Isn't this where the 'dirty work' of sharing takes place, rather than in any hypothetical 'CO-OP'?" This is no hollow software reuse program: It is people interacting directly in a largely apolitical and egalitarian community where corporate allegiances are left at the door.

2.6. Keeping Expectations in Check

From the earliest days, the pattern community adopted a posture of not promising too much. Many of the founders of the pattern community had been part of the rising tide of object-oriented methods, had made strong claims for objects, and had seen those claims dissolve or sink. Those same people didn't want to set themselves up for the same kind of failure again.

Therefore, a strong and explicit element of the early culture was to continuously bear a caveat emptor: to warn people that patterns were a new idea, that they were not a miracle, and that they only complemented other existing techniques. We hoped to prevent, or at least delay, the hype that had afflicted object orientation.

This value also served as one of a set of checks and balances that we hoped would keep the culture from getting out of control. By “out of control”, I mean a vague sense of severing itself from reality by becoming ingrown. Rather than holding the culture under a cultish set of externally imposed, absolute rules, this value encourages introspection and care on the part of its members. Such a posture can support healthy growth, the kind of growth that was uncharacteristic of new software ideas of the preceding decade. In fact, many of the values—those related to human dignity, to empirical grounding, and to systems thinking—helped keep the cultural processes open and progressive.

It worked for a while. The early pattern books were done in community, and reflected patterns that had come out of the PLoPs. Many articles about patterns warned of the dangers of hype and encouraged people not to abandon their good practices overnight.

Later books and articles would adopt the word “pattern” to lend credibility or power to ideas that perhaps otherwise would have had lesser or marginal value. These works didn’t embrace the tenets of the pattern culture, and at best would embrace one of the forms for writing patterns. At worst, some of them just adopted the name: pattern. However, the stipulation against hype seems to have limited the damage. Few pattern works, even today, make strong claims about productivity or success in the sense that the object-oriented community had in the past.

That doesn’t mean that expectations haven’t gotten out of line. Outside the pattern community, programmers frequently used patterns where other techniques would have worked much better. Too many programmers looked for patterns under every rock; this led to the later retort from the pattern community: “there is no prize for the most patterns.” This disconnect remains as a problem to this day.

2.7. Sources of the Culture

Though one can argue that the software pattern culture was a designed culture, an alternative view is that the cultural norms were borrowed and synthesize from other contemporary cultures.

The spirit of shedding academic pretense owed to the culture of the small but successful OT conference in the UK—a conference that had its own strong culture. Bruce Anderson was one of the founders of this event, and his influence strongly shaped the early pattern culture. Bruce was also a principal of the OOPSLA Architecture Handbook Workshop, an annual

gathering of designers who were striving to build a common body of design literature for the object-oriented community. The pattern community notion of building a common body of literature owes much to this vision. It was at that forum that many of the original pattern people would first come together.
The anthropocentrism of the object-oriented software culture was also a strong influence on the culture. The object paradigm had softened some of the overly formal and depersonalized notions of software development that came from common interpretations of software methods of the 1960s and 1970s. People thought of object-oriented designs as embodying anthropomorphic agents that carried out tasks inside a program to achieve some overall program goal. Each object designer put themselves into the role of the objects for which they wrote code. Some design activities such as CRC cards were themselves anthropomorphic exercises. Beneath this approach, one finds a devaluation of technological, tool-centered approaches and an embracing of human approaches. The pattern community carried forward and amplified those values.
Another hallmark of the object community was its grounding in industrial practice. Much of the growth in programming languages (such as C++) and object-oriented practice came not from academia, but from industry. This grounding in industry transferred into the pattern community as grounding in the experience of everyday programmers. The early pattern community perhaps recognized that most good software ideas had come out of industry rather than academia, and therefore viewed academics with suspicion, perhaps viewing them even as opportunists who embraced and advocated new technologies prematurely.
It is understood that Alexander's work heavily influenced the pattern community values: its focus on the human element, its notion of community, and its decentralization of authority. The community was selective in what it took from Alexander, and the selection process was arbitrary and probably not very thorough.

2.8. The Pattern Culture and Software Culture in General

The pattern culture, like all cultures, is based in a value system. Such values can be gleaned from the previous sections. Cultural values keep the culture cohesive and guarantee its survival as an entity, as a closed economy. One can view the above values as contributing to such a cultural identity and as securing the survival of the culture, if even for its own sake.
However, that was not the goal of the founders. The pattern culture is embedded in a larger enclosing culture. Kroeber (1948) discusses these levels of cultural patterns in his seminal work of anthropology. *Universal patterns* describe universal human behavior; for most programmers,

source code and an imperative style of programming characterize common trappings of culture. *Systemic patterns* reflect practices that owe to a common heritage; in software, the object-oriented community is rooted in common practices that go back to the early programming languages Simula (and its descendant C++) and Smalltalk. *Total culture patterns* reflect practices germane to a particular community: such patterns differentiate C++ programmers from Smalltalk programmers.

The pattern community was created as a culture within a culture. Created within the object-oriented culture, it was designed to serve and solve the problems of the culture in which it was embedded. Its original members were not only members of the object-oriented community, but in fact were leaders in that community. In retrospect, this was probably a key factor in the success of the pattern discipline. I return to this topic in Section 4.

2.9. An Anthropological Footnote

If one were an anthropologist visiting the pattern culture, what would one notice? Let's examine a couple of conventional hallmarks of culture: time, and the tradeoff between written and oral traditions.

Hall (1996) believes that time is one of the fundamental underpinnings of culture. Anthropologists classify cultures according to how they view time. Monochronic cultures believe that time is literal that that it adds up algebraically; German cultures can be said to be largely monochronic. Polychronic cultures, such as most Latin American cultures, treat the time of concept more loosely and have a higher degree of parallelism. Monochronic cultures tend to focus more on order; polychromic cultures tend to focus more on relationship and people. The pattern culture is difficult to classify as either monochronic or polychronic. Its Writers' Workshops are quite tightly scheduled on a small scale, and manuscript deadlines apply on longer scales. Yet, the community is highly social. One source of support for social activities is explicitly allocated unstructured time. This, too, may owe to the fact that this is a manufactured culture: it takes an extremely high degree of structure to give the feeling of unstructuredness.

On the surface, the pattern culture would appear to gather around a written tradition: the writings of Christopher Alexander or of the so-called Gang of Four (GOF) book (Gamma et al. 1995), or around the body of pattern literature that the rest of the community produces. Yet, the literature is targeted more for the market outside the pattern community than for the pattern community itself, and Alexander's works do not wield the influence of a written tradition. Many of the pattern community tenets related in this paper form only an oral tradition; this paper may be the first place they have found expression together in written form. Most software cultures are in fact oral cultures; the pattern culture builds on

this infrastructure of oral culture to infuse strong elements of written tradition.
In short, the pattern culture seems to defy easy classification along classic cultural lines. This richness of culture, and tolerance for the base features of vernacular culture, may be one of the factors that supported the spread and growth of this community.

3. Properties of the Emerging Culture

The values mentioned in Section 2 took root and stayed remarkably stable through the early history of the pattern community. To a large degree, the founders' vision worked. Most successful cultures evolve slowly around their roots, or even away from their roots. The pattern culture grew, and it learned as it grew. Some core values evolved and new ones came into play. Here, I summarize some of the key changes in the pattern culture over time.

3.1. Evolution of the Core Culture

A culture needs a place to live. The focus of the object-oriented culture might be said to be embodied in its main ceremonies: the annual ECOOP and OOPSLA conferences. Most values of object orientation were borne out at those conferences, and most local practices owed to ideas presented at those conferences. The "place" where the pattern community lived was at its conferences, called PLoPs (Pattern Languages of Programs), where authors came together for Writers' Workshops, interspersed with games, and fueled with social eating and drinking at a beautiful or distinctive meeting venue. There have become more PLoPs than OOPSLAs and ECOOPs: the pattern phenomenon had, and still has, more of a local focus. Furthermore, local pattern groups started springing up in cities and companies as forums for local pattern reviews preliminary to the PLoPs. The values and principles of the pattern community were most strongly embodied in the PLoP conferences, where attendees could count on the support of peers to sustain the unusual behaviors of community document review, games, and concern for human dignity. However, the distribution and breadth of the pattern community also caused these values to precipitate into local companies and groups. AG Communications Systems, a spin-off of AT&T and GTE, had its own strong pattern culture for many years (Rising 1998). Parts of AT&T Bell Laboratories, Lucent, and Siemens had strong pattern cultures through parts of the 1990s.
The pattern culture has evolved as it has grown, and much of the evolution took place at these conferences. New values refined old ones, and

new practices replaced or enhanced old ones. Some emergent hallmarks of the culture that appeared over the years include these:

- One is recognized for what one does rather than who one is. The people who sustained the strongest leadership positions (became conference chairs) and who gained some of the highest levels of respect in the community were those who wrote patterns or organized events, rather than those who brought innovation or ipso facto stature into the community.

- Pattern conferences came to be associated with definitive hallmarks and rituals focusing on good food and strong social environments. Each conference has its own personality and its own social environment. EuroPLoP features a late-night Stube culture; ChiliPLoP a cowboy-style dinner (with a real singing cowboy) under the stars arrived at by horseback. These personalities give each venue its own identity and perhaps prevent the pattern community from feeling like a franchise.

- The community started to move beyond individual patterns to embrace pattern languages. As the community explored the space of patterns early on, it was circumspect about its ignorance and about the need to learn and explore. As time went on, the community started identifying collections of patterns that worked together, and individual authors tried their hand at writing collections of patterns rather than individual patterns. Some of these collections probably comprise true pattern languages in the Alexandrian sense, while others are just topically related collections of patterns. This evolution speaks to the community's interest in addressing system-level problems in keeping with one of the original goals of the community. Norm Kerth in particular has been verbal about individual patterns being a dead-end street.

- The restriction against "going meta" gradually relaxed. The first steps included patterns about how to write effective patterns (Doble Meszaros 1998) and later steps would include deeper inquiries into the theory of patterns. Academia went through a flurry of work to formalize patterns—much of it disconnected from the theory of patterns and from the pattern community—but little of that work had much of an impact on the pattern community itself. One exception was Pree's book (1995) on meta-patterns, a book derived from his habilitation thesis work. Its patterns find occasional vernacular application among practitioners, particularly in Europe.

There were elements of the core culture that needed a place to live, but which never found one. Uncertainty arose now and again about how to enforce ethics, about defining quality standards, and about how to enforce or even discuss community-wide concerns. There was not even any central place to find out what patterns had been published, or even what ones had been presented at pattern conferences. Its stance as a shadowy cabal left Hillside without direct power to mandate practices at PLoPs or in the publication processes. It hadn't taken on even the basic administrative tasks of centralizing PLoP publications with the title Pattern Languages of Program Design, or PLoPD. (This also relates to the "somebody should" principle.) Because the pattern community had grown to thousands of people, it was difficult if not impossible to create a social process of dialogue for bringing those people together even to discuss crucial issues. This lack of centrality has arguably led to a weaker pattern culture today than it might have been with a careful balance of centralization.

Some core values have slipped away. Any initial focus on serving the customers of software systems has at best been tucked away as a tacit value, but in many contexts, this concern is lost. The initial pattern community built literature about the structure of programs, of object-oriented software. A few people wrote patterns about the structures of relationships of the people who wrote that software (e.g., Coplien 1995). However, few pattern authors talked much of the direct implications of software on daily human life; it was enough for them that so many human activities somehow depended on software, and that to make software good was somehow to make life good. Only the HCI community (see below), and to some degree the pedagogical patterns community, would embrace this perspective. This state of affairs probably represents a major opportunity for introspection by the contemporary pattern community.

3.2. Growth through Subcultures

Several special-interest groups split off from the pattern community to write patterns in individual domains. These communities "split off" in the sense that they met among themselves as a group, often in workshops or other meetings that were separated from other pattern activities. As such, these communities established their own identities and started creating their own bodies of literature. Two examples are the HCI community, one of whose early publications was (Borchers 2001), and the pedagogical patterns community, whose literature exists largely on the World-Wide Web (see http://www.pedagogicalpatterns.org). The HCI community is noteworthy because of its outward focus that is concerned with the quality of life of software users, rather than that of programmers. The software pattern community failed to embrace this early facet of the architectural pattern community in any tangible way. Other special-interest groups, such as those interested in organizational patterns, continue to work within the mechanisms and meetings of the pattern community.

3.3. Patterns and Vernacular Culture

Pattern culture norms reflect a collage of vernacular cultures. Language is a major component of culture. The question would arise: if we are building a community, should we have one language?

The issue never rose to any level of dialog in the community, and in most situations, English won out as default. There was some consideration of doing German patterns inside Siemens, but the desire to broadly publish works outside the company turned the decision to the lingua franca.

The story was different in Japan. Most Japanese patterns are written in Japanese by the Japanese and for the Japanese. Most patterns at the first Japanese patterns conference were in the native tongues of the attendees, though some had also brought along English translations. At one point, the goal was to publish a PloPD-like book containing both translations. At this writing, it appears that the main publication venue for these works will be the PloPD-5 book, in English.

Some companies sustain their own internal pattern cultures in their native language; I have noted this in Germany, Denmark, Japan, and France. In most other venues, English is the language of choice.

The same arises for programming language cultures. One low-level value of the community is that “true patterns” transcend programming languages. Language-specific patterns are accorded the somewhat “lower” title of *idiom*. This value was exemplified in the GOF (Gamma et al. 1995) book, whose patterns were written in a language-independent way, with specific examples in different programming languages. The language issue remains, however; many pundits accuse the GOF patterns as compensating for weaknesses in specific programming languages.

4. Outside View of the Culture

Cultures exist to solve problems. The pattern culture has value to the degree it solves problems—not its own problems, but those of the community in which it is embedded. The outside world has adopted several different views of the pattern culture. We might characterize them as follows:

- It is not a culture: it is a cult.
- Culture is irrelevant; patterns are about technical issues
- What culture? Literature from opportunists, from those outside the pattern culture who adopted the fashion of patterns, and in part from the community’s own GOF (Gamma et al. 1995) book is devoid of cultural allusions
- There is indeed a pattern culture, but they have it all wrong.

Here, we investigate these and other facets of how the outside world accepted, or not, the pattern culture.

4.1. The Value of the Pattern Culture

In Section 2.8, I noted that the pattern culture is embedded first in the culture of object-orientation, and more broadly in the culture of software development. The pattern culture is special in that it was created to help the cultures in which it was embedded, to solve problems within those cultures. The long-term goal would be to infect the enclosing cultures so thoroughly that the pattern subculture would lose its identity as a separate thing. At the early Hillside meetings, someone remarked that a true sign of success in patterns would be that people just talked in patterns, and that the word "pattern" would cease to be explicitly used. That would be the ultimate test of patterns' success in pervading the culture of programming.

In fact, it's worked out somewhat that way. The pattern subculture has maintained its identity in the "place" the culture carved out for itself early on: the PLoP conferences. These conferences continue to be the loci of the most focused practice and application of the tenets, values, and principles of the pattern community. However, the pattern culture strongly extended its influence to the mainstream events such as OOPSLA. If one strips out the technical papers from the OOPSLA program (a relatively small fraction of the overall conference) and considers the rest, its values and topics are difficult to distinguish from those at a pattern conference. OOPSLA workshops became less exclusive over these years. The Design Fest, a popular OOPSLA event pioneered by Ralph Johnson, reflected much of the open social environment of a PLoP conference.

Patterns become the lingua franca of high-level software architecture. Before the advent of patterns, the literature saw a recurring interest in software architecture definition languages. It is likely that patterns displaced these efforts, and provided an accessible, informal way to structure and disseminate the basics of software architecture structure. UML of course also became popular, and while it occasionally is used to convey high-level architecture notions, it is more commonly relegated to a position as a kind of graphical C++, with its domain being medium- and low-level design. Many major architectural styles took patterns as their main form of expression. Trygve Reenskaug started using patterns to describe the model-view-controller (MVC) architecture. Distributed systems architectures in general, and broker-based architectures in particular, featured pattern literature as their primary form of design documentation. Advanced programming techniques became encoded as Java patterns (a quick survey discovers ten books that could be called

Java pattern books). Fowler's Analysis Patterns became one of the foremost references on analysis techniques (Fowler 1996).

The PLoPs also served as archetypes of a new trend in software conferences: small, topically focused conferences that stood in contrast to the large conferences sponsored by professional organizations. These small conferences ran on smaller budgets and offered more concentrated, industrially relevant value than the larger, academic conferences. As companies started tightening their belts in the late 1990s, these small conferences would become increasingly popular while the larger conferences would languish. It is difficult to say whether the PLoP conferences invented this trend, but they were at least an early, large and visible part of this trend.

Even within established computer science conferences, one finds trends either that owe to the pattern community traditions or which evolved in parallel with those traditions. One example is the growth of shepherding activities in professional conferences. Academic conferences traditionally accept a manuscript and either accept or reject it on its merits with no further opportunity for interaction. Conferences like OOPSLA have started taking papers that have strong technical merit but which need improvement in their expressiveness, and instituting shepherding activities for those authors. A program committee member works with the author to improve the work in a time-boxed shepherding activity. The work may be accepted or rejected at the end of that period, based on the recommendation of the program committee member and the decision of the program chair. Most of these exercises have succeeded in improving the paper to an acceptable level. I remember Adele Goldberg as being an exemplary reviewer and shepherd as early as OOPSLA 1996.

We can categorize these impacts as cultural or stylistic. For example, American culture is influenced by English culture in several different ways: Americans adopt many English culinary practices (e.g., eating meat and potatoes) as true elements of systemic culture, while adopting others (e.g., miniskirts) at the level of style or fashion. The pattern culture influenced software development at both of these levels. For example, many of the pattern books (e.g., the Analysis Patterns book) do not explicitly build on the ideals and principles of the pattern community, though they use the pattern form: the literary format used to express patterns. That is an influence at the level of fashion.

Did patterns have any real cultural impact on software development? The sources of true cultural change are difficult to trace, so it is difficult to be conclusive in this regard. It is noteworthy that the OOPSLA conferences took an increased interest in human issues after Alexander's appearance at OOPSLA in 1996: more keynotes, panels, and workshops focused on the human issues and less on the hard-core technical issues than they had in the past. The Software Developer conference, for example, added entire tracks for human-related topics. Program committees grew to embrace

shepherding. The pattern literature became citable, even by academics, and there was an explosion in pattern-related thesis topics in academia.
Furthermore, the culture successfully achieved some of its original objectives in everyday software development. Programmers started using patterns in their design vocabulary. Methodologists started using patterns to describe their work, and programmer tools that came from methodological strongholds like UML found it necessary to express things called patterns. Moreover, speaking of "body of literature," many of the best-selling computer science books were explicitly about patterns.
Whether both the pattern discipline and the industry as a whole, including venues such as OOPSLA, rose with the same tide, or whether mainstream conferences built on the pattern foundations, is hard to tell. What can be said is that the pattern discipline was at the epicenter of a cultural shift that took place in software in the 1990s.

4.2. Cult or Culture, and Academic Disdain

The pattern community became a vibrant but highly normative and highly stylized culture in the mid- to late-1990s. The original intent of the culture to be a little bit shocking to the academics who might approach it out of curiosity or desire to engage it sometimes worked too well, and the culture gradient between academia and the pattern community sometimes became a problem.
One of the problems was in publication—a key activity and raison d'être of the pattern culture. Because of the expertise inversion practiced in the pattern conferences, a pattern publication bore none of the hallmarks of originality or of top-down control that one found in academics. The academic view was that pattern publications were not subject to adequate expert scrutiny or quality criteria. Academics assembled committees of experts whose judgment maintained the long-term quality of their bodies of literature; we had no such committees. But the key consideration was that publication serves as a key indicator of success, prestige and accomplishment in academic culture; the exclusive review process that caused academics to ascribe value to their publications. Such review made such publications a scarce resource. The other factor that made academic publications a scarce resource was the need for originality: it was permissible for an idea to be published only once. Publication lists were a substantial consideration in granting academic tenure or professional promotion in research laboratories.
The pattern community adopted different values. We did not value novelty; we instead valued applicability. Applicability implies reproducibility, and in this sense, the pattern community was closer to the academic value of scientific inquiry than either community realized at the time. The pattern community also honored prior art and exhorted pattern

writers to build on and cite prior work; in this sense, the two communities were also similar. The pattern community stipulated a stringent review process that started with peer supporting called shepherding, that took the author through a *public* review of their work, and which culminated in a stringently edited publication process for the PLoPD books (e.g., Coplien and Schmidt 1995). Pattern works are more closely scrutinized that most works in academia. In the end, the communities divided along the lines of originality. The uniqueness of academic publications was what gave them their value; the pattern people had an "aggressive disregard for originality."

Because the pattern culture and its members were embedded in the larger object-oriented community, part of which comprised academics, the publications overlapped. If publications were the currency of academic accomplishments, the pattern people were printing money. The academics viewed them as counterfeiters.

The perspective was exacerbated by other pattern community values. The pattern community was a counter-culture, and that was off-putting to those whose identities or careers owed much to the old culture. Many couldn't see the value of games and found them gratuitous: a kind of sick dandyism that didn't serve the industry at all.

The question started to arise as to whether the pattern community had become, or in fact originally was, a cult. It was a strong charge. Whether the pattern community could be viewed as a cult hinged largely on how one chooses to define 'cult.' The question led to isolated introspections among pattern community members about whether the community had gone too far.

Part of this question revolved around Christopher Alexander whose works had given the community much of its inspiration. If the pattern culture has a Shaman, it is Alexander. Many of the community values in fact derive from his work and writings. Taken together—which is difficult, as his writings are voluminous and sometimes difficult—his works represented an extreme counter-culture. The software pattern culture reflected only the tip of the iceberg of Alexander's extreme views. Some of Alexander's tenets were so extreme as to be exclusive, and this impression carried over to the software pattern community. The pattern community was an extreme community, did not hold to prevailing convention, and was guided by the teachings of an authoritarian figure, albeit a somewhat unwilling one. This alone led some to incorrectly criticize the community as a cult.

The pattern community has striven to avoid cultism by striving to be both an open and inclusive community. The conferences and other activities of the pattern communities are open activities; its literature is an open literature; its membership is international and crosses technological culture boundaries (such as programming language orientation). No single pattern form is mandated. Early authors were encouraged to experiment with pattern form, out of a realization that we lacked a broad base of experience on which to dictate such matters. As the community has matured, there has been a growing preference for the so-called Alexandrian form.

As mentioned briefly in Section 2.6, the values themselves worked against cultism. The values of human dignity, empirical grounding, and systems thinking helped insure that the cultural processes would be open to observers and neighbors of the culture, and that the culture would not become an isolated cult. It is this set of checks and balances that have likely served to keep the pattern community values as stable as they have been over the past decade. A cult is a closed system and cannot be healthy; the pattern community, as a culture, interacted enough with the outside world to enjoy reality checks.

One element of the pattern community was viewed as secretive: the Hillside Group itself. The best way of describing Hillside might be as a support group for its members, members who worked for the most part alone or in small groups as change agents to usher patterns into the software world. There were a few things that the group discussed and supported collectively, such as the PLoP conferences, but it acted behind the scenes in doing so. Furthermore, these activities quickly fell into administrative routines that took little of the energy and focus of the Hillside membership.

In fact, the early Hillside members often discussed their modus operandi: whether to operate broadly in the open or to stay behind the scenes as a "shadowy cabal." The "cabal" model won out. The rationale was that the community should establish its own identity independent of the founders collectively, though it might build on the efforts and agendas of the founders individually. The body held control over sponsorship of PLoP conferences and over choice of editorial personnel for the PLoPD books in a managerial capacity. Because the books and conferences were the most visible outward manifestations of the pattern community, and because people knew that this thing called Hillside existed, and because Hillside acted as a cabal, people quickly took the notion that the community was being run by a closed process. The control was much less extensive than anyone could imagine, but the perception persisted. In recent years, Hillside membership has grown beyond a dozen people to dozens of people; the definition of membership has become more ambiguous; only four of the original "cabal" remain; and the group comes together in open forums (usually associated with OOPSLA as an annual event). These moves toward openness seem to have dispelled the notion of Hillside being a "shadowy cabal."

It came as no surprise that few high academics sought publication at early pattern conferences. The pattern community would later look at its own values and try to accord inclusiveness to academics as well, but found themselves at a loss to offer the academics things they would value. This discord, however, was minor and relatively short-lived. Patterns had taken the software community by storm and, academic reservations not withstanding, patterns were having broad influence. They had become a vehicle for grass-roots influence and, against the desires, of the founders, had become a movement and a fad. The publication support of patterns and their influence first on industrial software design—largely through such works as the Design Patterns book (Gamma et al. 1995)—gave them an air of novelty. Academics could now take license to explore patterns as something that was both relevant and, ironically enough, new.

4.3. Culture versus Progress

One purpose of a culture is to maintain stability. One should be able to wake up in a culture every morning and expect the same rituals, language and values; these invariants help people interact with each other efficiently. Cultures also give people a sense of belonging. These deep human traits work against change in culture.
As the pattern culture has grown, members of the culture have hung on to many of the original traditions, often out of touch with the reasons for their institution. That has sometimes made it difficult for the pattern community to grow. For example, one early piece of pattern literature was the so-called GOF book (Gamma et al. 1995) that offered a set of micro-architectures under the pattern label. The book was one of the first publications of a small set of useful related patterns and became a commercial success. It had broad influence. From a cultural perspective, this book can probably be thought of as the most canonical of the pattern literature outside the pattern community itself. (The canonical works inside the pattern community might be said to be Alexander's books, but the boundary between these two communities is vague and the influence of the respective bodies of literature difficult to assess.) The GOF book says very little about pattern languages, and explicitly claims not to be a pattern language. It has been difficult to grow the pattern community into pattern languages, arguably in part because of the influence of that work.
Part of this stability may in fact owe to the culture's origins, which feature an interesting paradox. Though the community advocated an "aggressive disregard for originality," it gained notoriety because of its differentiation from the status quo—it was the epitome of novelty and extremity in what had become a routine and mundane world of object-oriented programming. Its norms, to some degree, were a guard against returning to the perceived evils of the status quo as rooted in academics and in methodology. As David Weeks points out, the software pattern community is in that sense very concerned with newness. Within that framework, any newness that changes the original newness threatens to restore power to the very constructs patterns had set out to obliterate, creating a sort of identity

crisis. Perhaps the community doesn't change because of its need to sustain the newness it created.
The pattern community has nonetheless evolved. In recent years, the review processes for pattern publication have become more stringent (pattern conferences now have a program committee). The community has given up its initial fear about "going meta" and has embraced studies of pattern foundations. The community has also spun off several sub-communities that have moderate coupling to the rest of the pattern community as a whole, including a community gathering human-computer interface patterns, another gathering pedagogical patterns. It is likely that each of these communities has its own culture (in the sense of local customs or total culture).

5. Conclusion and Acknowledgments

By all outside appearances, the Hillside group planted the seeds of a culture that grew and thrived while maintaining most of its core tenets over more than a decade: an eternity in Internet years. Is the process repeatable? It is a difficult experiment to repeat. Human behavior is uncertain enough, and context in general unpredictable enough, that we leave you with the admonition: don't try this at home.
It is difficult to say whether the pattern discipline led the cultural changes we have seen in software over the past ten years, or whether the pattern values were simply part of the broader patterns of change of the same period. It is even difficult to say whether the alignment of the culture with the wishes of the Hillsiders was simply the result of chance, abetted by tacit foresight of where the industry would head. In fact, it is likely that the pattern discipline and its principles simply created a well-formed catalyst that reflected emerging values of the time, a catalyst that precipitated the new culture.
It doesn't matter. What is important now is to understand the pattern values, norms, practices, tenets, and mores as they exist and where possible to exploit them to improve the quality of life for the constituency served by the software craft, and to improve the quality of life of the programmers at the core of this craft. Those were the initial goals of the Hillside Group that came together to create this culture.
Many thanks to Ken Auer, Gertrud Bjørnvig, Alistair Cockburn, Ward Cunningham, Cecilia Haskins, Luke Hohmann, Ralph Johnson, and Linda Rising for comments on the paper. A special thanks to David Weeks for particularly thoughtful comments that shaped my own thinking about these issues.

6. References

Alexander, C. (1979). The Timeless Way of Building. Oxford: Oxford University Press.

Borchers, Jan (2001). A Pattern Approach to Interaction Design. Chichester, UK: John Wiley and Sons.

Coplien, J. (1992). Advanced C++ Programming Styles and Idioms. Reading, MA: Addison-Wesley.

Coplien, J. (1995) A Development Process Generative Pattern Language. In Coplien and Schmidt (1995), 183—237.

Coplien, J. and D. Schmidt, eds. (1995). Pattern Languages of Program Design. Reading, MA: Addison-Wesley.

Dictionary: dictionary.com, accessed 1 May 2004.

Doble, James, and Gerard Meszaros (1998). *A Pattern Language for Pattern Writing*. In R. Martin et al., eds., Pattern Languages of Program Design — 3, Reading, MA, Addison-Wesley.

Fowler, Martin (1996). Analysis Patterns. Reading, MA: Addison-Wesley.

Gabriel, Richard (2002). Writers' Workshops & the Work of Making Things. Reading, MA: Addison-Wesley.

Gamma, E., R. Johnson, R. Helm and J. Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley.

Hall, Edward T. (1996). The Dance of Life: The Other Dimension of Time. New York: Peter Smith Publications.

Hohmann, Luke (1996). The Journey of the Software Professional: The Sociology of Software Development. Upper Saddle River, NJ: Prentice-Hall.

Kroeber, Alfred L. (1948). Anthropology: Culture, Patterns and Process. New York: Harcourt, Brace and World.

Pree, W. (1995). Design Patterns for Object-Oriented Software Development. Reading, MA: Addison-Wesley.

Rising, Linda (1998). The Patterns Handbook: Techniques, Strategies, and Applications. Boston, MA: Cambridge University Press.

Jim Coplien is an international consultant and author of several books, including the critically acclaimed "Organizational Patterns of Agile Software Development," and is best known for his seminal work on C++ idioms, multiparadigm design, and organizational patterns.