Patterns and their Umwelt: How to Make Patterns Come to Life

REBECCA WIRFS-BROCK, Wirfs-Brock Associates, USA
CHRISTIAN KOHLS, TH Köln, Germany

Design patterns promise reusable solutions to recurring problems, yet many remain unused, misunderstood, or applied mechanically. This paper argues that patterns often fail because they insufficiently account for the umwelt—the situated perceptions, experiences, intentions, and values of authors, designers, users, and the contexts in which patterns are enacted. Building on Christopher Alexander's notion of living structure, we reconceptualize patterns as potentials rather than finished solutions: seeds that must be unfolded through situated design decisions. We identify common "pattern smells" that hinder effective use and show how attention to unfolding, roughness, and contextual fit can make patterns more resilient and meaningful. Extending this perspective, we emphasize the importance of vividness in pattern descriptions—clarity, conciseness, and expressive forms of representation that engage designers, invite reflection, and support learning across diverse umwelts. We conclude with guidance for writing, maintaining, and evolving patterns so that both their descriptions and implementations remain alive, adaptable, and capable of resonating with changing environments and practices.

Categories and Subject Descriptors: Software and its engineering~Software design engineering • Software and its engineering~Software design tradeoffs • Software and its engineering~Design patterns

General Terms: Software Design

Additional Key Words and Phrases: Patterns, Design Theory, Christopher Alexander, Participatory Design

ACM Reference Format:

Wirfs-Brock, R. and Kohls, C. 2025. Patterns and their Umwelt – How to make patterns come to life. HILLSIDE Proceedings of 32nd Conference on Pattern Languages of Programs, People, and Practices (PLoP). (October 2025), 15 pages.

1. PATTERNS IN CRISIS

There are many patterns out there. And many are quite successful. But, we also see patterns applied the "wrong way." There are certainly patterns that do not make designs better, but make them worse. Moreover, there are many good patterns that are not used, either because they are not discoverable, or not interesting, or not easy to understand or implement.

We enjoy writing patterns. We enjoy reading and thinking in patterns. We enjoy implementing patterns and seeing how they unfold and bring benefits to our designs. However, there is an enormous body of patterns that are never used. They are workshopped, reviewed, and published. But rarely used. They are not referred to in further publications, neither in academic papers, practice-oriented articles, or code—or any other material using patterns. We don't learn about their success in practice—the "Applied Patterns" track at EuroPLoP was such an attempt—and we rarely see revisions let alone critical examinations of the claims in the patterns. Hence, there is a strange feeling that not all patterns work, even though it appears that nothing is obviously wrong with them.

At the same time, we see collections of patterns that are quite successful in practice. There is of course *A Pattern Language* by Christopher Alexander (and his colleagues!), the famous Gang-of-Four patterns about software design, and many domain-specific books about patterns. So, there seems to be some aspects of those particular patterns that make them more useful, more successful than others.

Sadly, even if patterns are used and applied, they are not always a success. Many things can go wrong: one can select the wrong pattern, one can implement it poorly, and one can follow a pattern too literally without taking into account the actual environment. Patterns are a means for designers. As such they can help and guide in the difficult process of making design decisions. However, they do not relieve us from making those decisions, making necessary adaptations, and taking responsibility for using them appropriately in our design process.

In this essay we will introduce the thesis that patterns—or pattern descriptions, rather—often do a poor job of accounting for the designer's and user's backgrounds, intentions, experiences, and specific situations. That is, for the authors', the designers' and the users' umwelts.

The word “umwelt” in German means “environment” or “surroundings.” We humans, like all living things, experience life in terms of our unique perceptions of our environment. What we perceive as notable in any particular context depends both on our innate capabilities to sense our surroundings and what we have learned to recognize as meaningful. Consequently, our actions are constrained by our perceptions of what we view as significant, and what we deem to be appropriate actions to take. Our individual umwelts are distinct.

We will show how an awareness of the authors’, the designers’, and the users’ umwelts can improve the quality of pattern documentation, leading to more vivid patterns, better implementations and usages.

We will first discuss what patterns are, why patterns are great and why they don’t always work. We draw from the philosophical works of Christopher Alexander as well as the insights and observation of the experienced pattern community (Hillside and Hillside Europe) and other researchers. Then, we will explore the concept of umwelt and what it means for patterns to be alive. Based on these insights we will identify an initial set of what we call “pattern smells.” The term “pattern smell” is borrowed from the software domain where “code smells” indicate that something is rotten in the code, alarming you that something is wrong[1].

Finally, we will draw conclusions on how to better account for the umwelt in pattern descriptions and bring the patterns to life in the individual contexts.

2. WHAT ARE PATTERNS, REALLY?

The term “pattern” has multiple definitions. Even within our pattern community, we have different meanings for it. It is therefore helpful to clarify these different meanings and precisely identify what we mean when we speak of patterns. In the most general context, a pattern is something that repeats. Consequently, because it repeats, we are likely to recognize (or perceive) it as having coherence. It is a recognizable “thing” that we can name and describe. So, for example, we might identify the patterns of leaf structure in a particular plant species, and speak of “maple leaves.” Patterns in both our natural and our constructed (designed) environment can be more or less discrete and temporally distinct (Weiss, 2025). We can speak of yearly rainfall patterns or voting patterns. We can speak of geometric patterns found in both natural or human-made things, for example crystals or woven carpets. According to Wikipedia, “A pattern is a regularity in the world, in human-made design, or in abstract ideas. As such, the elements of a pattern repeat in a predictable manner.” But what is it about patterns in our designed (constructed) world that distinguishes them from any other repeating structure?

2.1 The Purpose of Patterns

Alexander offers multiple definitions of what patterns are. The most well-known is: “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.” (Alexander, Ishikawa, Silverstein, Jacobsen, Fiksdahl-King, Angel, 1977, p. X) Moreover, he points out that patterns are both a process and a thing (Alexander, 1979). There is a specific repeatable sequence that leads to the recurrent structure.

A pattern description should capture the recurrent context, problem and forces, solution form, the recurrent process to create the solution and the recurrent consequences. Sometimes we refer to a pattern description laxly as “the pattern” when we actually mean “the pattern description” or “the pattern described here.”

To consider patterns as recurrent forms is a definition without judgment. However, usually we want to share patterns that are good; that is ideal forms with ideal consequences. Another common meaning of pattern which has an overlap with how we in the patterns community define patterns is: “a form or model proposed for imitation; an exemplar.” The German word “Muster” (pattern) also conveys the meaning that the pattern is some form of ideal or a good model: a Musterstudent is an ideal student, a Musterorganisation is an ideal organization. According to the Oxford English Dictionary, the origin of the word pattern is rooted in “patron” meaning “something serving as a model”; a “patrone” was a “work by exemplar.”

Departing from a single reference example, “pattern” means now the original form of instances. This good form, the ideal reference model is somewhat like an ur-form, an ideal or form idea. An idea, however, is not the thing in the world. And a pattern transcends all instances of that idea. It is a potential that needs to be brought to the world. This coming-into-being is the process of unfolding the pattern.

[1] https://wiki.c2.com/?CodeSmell

Because the salient aspects of a pattern are recognizable, with enough experience in seeing significant patterns we can predictably identify them. And as designers, we can apply those patterns to our own creations—and spot them in other designs.

Being ideal, a pattern should eventually improve the situation of a system. It should have a positive impact and make the system more robust and resilient. A good quality measure is to consider what is gained by the pattern. Does it simply solve a problem or does it also add harmony and beauty to your code? Does it make your system more flexible and more open to adaptations? Does it make it more resilient? Does the solution interact more openly and reliably with its environment? Does it make you feel more alive or comfortable if you look at the code? The impact of a pattern is bound to a point of time—new technology might change the value of a solution. Hence, a pattern should also be open to replacement. If your system starts to rely on a pattern and gets dependent on its specific solution, then its life may not be sustainable. We speak of "dead code," "end of life," and "software lifecycles" for a reason.

A good pattern can and should increase the vitality of a system, making it open to future revisions and adaptations.

2.2 Patterns are great!

"The specific patterns out of which a building or a town is made may be alive or dead. To the extent they are alive, they let our inner forces loose, and set us free; but when they are dead they keep us locked in inner conflict." (Alexander, 1979, p. 101).

Capturing good patterns and describing them in written documents offers many benefits. The goal is to search for patterns that make a place or a design more whole, more alive.

We all know situations and places where we feel free and comfortable—the coffee shop around the corner, a cozy chair by the fireplace on a cold winter day, an office desk at the window. We have seen code that is beautiful. We remember lessons in school that brought enlightenment, situations where we got sucked into a moment of flow, and discussions that made us feel alive.

The configurations that raise such positive feelings in humans are good solutions, they are the patterns we are looking for. They have an organic order, "the kind of order that is achieved when there is a perfect balance between the needs of the parts and the needs of the whole" (Alexander, 1975). Good solutions harmonize with human needs. Software code that has been optimized for mathematical needs only, rather than human understanding, can hardly be considered good as it is harder to maintain, more error-prone and uneasy to "live" in (Gabriel, 1996). And yet, software code that has been optimized for programmers' needs might fall short of meeting its users' needs.

Besides this striving for beauty, harmony, wholeness, a holistic quality without a name, and life (whew!), patterns are meant to solve serious and difficult problems. Very often we are not even aware of the problems that cause our miserable feelings. A conference without well-planned coffee breaks ignores that attendees not only need a rest between the presentations and workshops, but also time to reflect and chat about the sessions in a comfortable environment. A programmer who designs a system that is not flexible enough to accommodate future adaptations, will encounter problems when the requirements change. Without the awareness of such serious problems, it is impossible to do something about them.

Experts very often know good solutions but are also more aware of common problems. They should make their design heuristics available to others (Wirfs-Brock & Kohls, 2019). Patterns are a way to express such tacit knowledge.

Design patterns are a toolbox from which a designer can choose the right tools. To reuse successful designs does not mean that one just puts together ready-made components. Patterns are not merely additive, but also are multiplicative in the ways they are combined. They are structural regularities that can overlap and that can be merged with each other. While design patterns are, "an aggressive disregard of originality" (Foote, 1997) in that they only capture existing solutions, they are by no means an edict against innovation. The combination of reliable forms that contribute to the larger whole opens the door to infinite new ideas by means of composition.

In this sense, patterns can also help to reduce the complexity of design tasks. A complex system is split into smaller parts that can be addressed somewhat independently. Those smaller parts are loosely coupled to each other, but they are not isolated because the larger context is always part of the analysis and design of the smaller part. A pattern language connects solutions and provides designers with a basic vocabulary for designing larger systems. It also provides guidance on which patterns work together and how a network of patterns can be used to build larger systems.

So, patterns are a great success story and they should be. They are applied now in many different domains, including architecture, software, human-computer-interactions, society and organizations, business processes, pedagogy, and many more. Still, they don't always work.

2.3 Patterns don't work

Patterns have their roots in building architecture. But not very many architects use patterns. In fact, many architecture theorists have criticized patterns for being mechanistic, conservative, and offered as solutions without supporting empirical evidence of their benefits. These criticisms often refer to the patterns of *A Pattern Language* (Alexander, 1977) and not so much to the design theory behind patterns. Alexander himself was never satisfied about how those patterns were used. In a later foreword to *Notes on the Synthesis of Form* (1971 edition), he laments that too much focus has been put on the method (the systematic way of finding the solution) while neglecting the inner quality of a solution and the resonance between form and people's perception of it.

The concept of patterns is central in *A Pattern Language* (Alexander, 1977), *The Nature of Order* (Alexander 2002a, Alexander 2002b), and *The Oregon Experiment* (Alexander, 1975). However, in his later work, the four-volume series *The Nature of Order*, Alexander rarely uses the word pattern. Instead, he speaks of centers that are more or less alive. In general, such centers increase the overall quality of a system (making it more whole, more alive). Patterns are meant to create such centers, i.e., they are specific solutions that create strong centers. Alexander (2002a) observed that the use of patterns often failed to achieve their potential because architects merely applied patterns without adapting them to their specific situation.

Another example: The domain of education and pedagogy has been capturing and sharing patterns for 30 years now. But in practice, many of those patterns are rarely used. Patterns in this area are sometimes hard to understand, they lack the gestalt—the perceivable form—and they are hard to recreate. Also, many pattern descriptions are just too long and boring, as they explore solutions in detail that could probably be better described with a short paragraph.

One of the authors of this paper has written many educational patterns himself. Each pattern has many forces and detailed solutions. While the forces build tension and reflect on many perspectives, they are boring to read! The classic description form, even the sequence of sections, seems to be wrong for such patterns. They are not concise. Their meaning gets lost in an avalanche of words.

It is in computer science, namely in all fields related to software design, where patterns have been most successful. Today, almost everyone involved in software knows that patterns exist, they roughly know what they are (even if only a few know about the full theoretical background, they often even know a number of patterns and have implemented some). It seems that everything is fine… But that is not the case.

It turns out that most software developers know fewer than 10 or 15 patterns. And there is an almost religious belief that using patterns makes software better by design. However, using the wrong patterns or using them improperly can cause serious harm. Too many patterns, ill-applied, can lead to over-engineered systems that are harder to maintain.

There is also a lack of empirical evidence and normative justification for most software patterns. While we expect some real-world examples for each pattern (sometimes at least three instances), we rarely test the assumptions captured in a pattern. That is, we do not verify the details of the referenced solutions or test whether the pattern works in the described context, nor do we test whether the forces actually exist as claimed in the pattern description (Kohls & Panke, 2009). The known-uses in a pattern description may be evidence for the existence of the pattern. But there is no validation that a pattern description captures the "true" essence of these examples. Furthermore, there is no evidence that the claimed interaction of forces, the cause-and-effect of context -> problem and field of forces -> solution -> consequences, is stable and tested. Also, empirical content is not enough for patterns: we are not just interested in solutions—we want to see good solutions. And that is a normative judgment.

Unquestioned values, a misunderstanding of empirical evidence for patterns, boring or difficult-to-understand descriptions, a forest of too many patterns, and overconfidence and trust in patterns, are just a few of the reasons why patterns can fail to work! Still, each pattern offers a great potential to improve our umwelt. We believe that many shortcomings of patterns are related to a lack of understanding how the umwelt of authors, readers, designers and users impacts the quality that is entailed in each pattern.

3. THE POTENTIAL OF PATTERNS

Patterns should not be seen as tricks. They are not ready-made solutions. They require the heart and the character of designers to bring them to bear in their designs. Designers who take the pattern and make it happen, using their own skills, intents and potentials.

Patterns are potentials as well. Some idea that has been seen to work in the past (or at least should work in theory), that can come to life through the unfolding by a designer.

A good pattern description accounts for the unfolding rather than the final structure. It helps to generate the solution. A pattern description should help designers to unfold their solution, to use the potential of the pattern and the designer's own potential. It should guide the decision making and be open for variation, while allowing for a certain level of roughness—rather than promoting static, polished out-of-the-box solutions.

Roughness (Alexander, 2002a) accounts for the special needs of a local situation. One finds roughness if parts are locally adapted rather than using pre-made modules that are plugged together like Lego bricks (Quillien, 2008). Wirfs-Brock (2010) interprets roughness as code or an implementation that "exhibits some aspect of hand-crafting to fit into its environment." Instead of using pre-made software components, code needs to be unfolded to the specific human, organizational and technical requirements.

Roughness is a consequence of unfolding and generating solutions in real world situations—the messiness of life. Hence, systems with roughness resonate with us. They feel real and authentic. Roughness makes systems more robust and resilient rather than fragile. Roughness invites change, it allows us to tinker with system components and improve them. With roughness, we designers can put our character into the system. Creating signature code, signature buildings, signature conferences.

To see a pattern as a potential, means to see it as a rough idea. A potential to be realized and implemented, either through thought experiments (thinking and planning of new cities, neighborhoods, landscapes, social affairs, even politics) or by actual implementation (building the cities, running the neighborhoods etc.). And often both. Even if you only think in patterns, you can have an impact on others who build; and if the builders think in patterns they will build differently. This is because a good pattern account not only talks about its (recurrent) structural appearance and its recurrent structural creation, but also about its (recurrent) decision making as you consider the forces, and its (recurrent) consequences as you evaluate the pattern's impact—positive, negative, neutral.

Both forces and the solution require a deep understanding and design skill. The designer can take the solution as an inspiration, as a guide to reflect and make justified decisions. Still, it is her responsibility to make full use of the pattern's potential. The character and the experience of the designer matter, but it is her umwelt that makes the pattern happen.

4. WHY UMWELT MATTERS

In *Discovering Your Software Umwelt*, Wirfs-Brock et al. (Wirfs-Brock, 2024) assert that "Our umwelts are multi-faceted. People operate in many contexts, such as home life, work life, and public life, where we perceive and act upon different things. For each of these contexts, we have our own specialized sub-umwelt." Furthermore, "[w]hile our umwelt dictates our perceptions and constrains our actions, it isn't static. Through interactions with our environment, we learn and adapt, and as we do, our umwelts grow and evolve with us."

The unique umwelts of pattern authors and potential pattern consumers have been shaped by their experiences. How, then, can pattern authors effectively share their patterned experiences with others? The challenge is to convey both their patterns' significance along with the essence of the solutions they purportedly solve. Pattern descriptions could and should be a bridge between these different umwelts.

4.1 The author's umwelt

As pattern authors, we bring our umwelt to bear when we identify and write about new patterns. Arguably, what's meaningful to us is a judgment call based on our unique points of view. While we are likely aware of other solutions to the problems we are concerned with, the patterns we select to focus on, and then subsequently name and describe are those we deem significant. If we identify more patterns and organize them into a collection or pattern language, we want those patterns to fit together. So, in our process of selection and coherence making, we leave out or discount whatever doesn't fit easily into our umwelt.

The idea of identifying a worthwhile pattern or forming a pattern collection feels subjective because it is subjective. What we pattern authors deem to be significant is a matter of our unique perceptions and tastes. What other designers deem to be significant is a matter of their unique perceptions and tastes. Subjective experiences and perceptions are all we can rely on. As designers and patterns authors, our umwelts have been shaped by our past subjective experiences.

Is subjectivity a bad thing? Well…no. A pattern doesn't embody an unachievable ideal that everyone should agree upon as universally good. As pattern authors, we share in our patterns our subjective views on how to create good structures. We write about structures that we have noticed and found useful in our environment that we believe also have the potential to work elsewhere. That's all we can do.

Both the ground for mining (creating, discovering) patterns and the chosen method for doing so have implications for the utility of a pattern. There is a difference between a single author reporting his or her patterns and the outcomes of a group discussion. Likewise, the range of domains and contexts in which a pattern has been observed is critical to its general applicability. For example, if a pattern has been observed multiple times in Java programs, does this mean that it will work for C++ code as well? Without having observed or tested it, one cannot really—empirically—tell.

We can hardly speak of objective or universal patterns as they are indeed found based on subjective or inter-subjective observations. The traces of perception, experience and manipulation of mental structures are a unique history for each individual, they therefore result in personal views of the world. A pattern description makes individual judgments explicit and therefore open for inter-subjective discussion and evaluation.

If some objects have little variation, e.g., soda bottles, it is easier to denote a form category people agree on. With a lot of variation, it gets much harder, e.g., a lecture can have many forms. People not only have different ideas about what an ideal lecture looks like, they might even debate if an extraordinary lecture is a lecture at all.

The functional forces documented in a pattern description are even less objective because different individuals may prefer different functions and have different values. Which functions are critical and relevant depends on the interests of the designer and their design context. The least objective properties are those of beauty. Arguably, there are objects that most people consider beautiful, e.g., rainbows or water lilies. But just as likely, they might consider a picture of these objects as mere kitsch.

In the aesthetics of Kant, Hegel, Wittgenstein and others (Majetschak, 2007), beauty and wholeness depend on our familiarity with a category. Somebody who is competent in a domain is capable of giving reasons for their aesthetic judgments. For example, we cannot only say that an OBSERVER is a beautiful software design solution in specific contexts, but also give reasons why it is the right thing. It is not just a solution but a good solution. A beautiful solution is one that does the right thing and fits well with our personal aesthetic categories. In the case of the OBSERVER, being a programmer is necessary to see the inherent beauty of this pattern. But because of our subjectivity, still we may not always agree.

Yet there are conventions that are implicitly agreed upon in a community of practice. Teaching practice and apprenticeship, as well as software design patterns make these explicit, and thereby inter-subjective agreement becomes more likely. Many norms in software design appear to be reasonable conventions, e.g., maintainability, robustness, performance, memory optimization, etc.

Different authors may associate different meanings and significance to the patterns they perceive. But it's not only the author's perspective that matters. It is also the umwelt of a designer that uses and recreates a pattern.

4.2 The designer's umwelt

Patterns, or good pattern descriptions rather, describe both a thing and a process; e.g., a way to unfold a solution for creating that thing. To Christopher Alexander (2002a), any process for creating living structures involves unfolding, a sequence of transformations acting on the whole. Henrik Karlsson (Karlsson, 2024) characterizes unfolding as a tight feedback loop between the designer and design context: "By gradually adjusting the thing you are designing and observing how well it fits the context, you create a feedback loop that embeds the context's knowledge into your design." Furthermore, he posits that our design context holds more nuance and information than we can keep track of in our heads. And that by employing a design process that enables us to incrementally extract relevant and timely information from our context, we get better design solutions. Especially if we use what we've learned from making one step to refine our next step. Unfolding is an intense process!

As software developers, our programming habits, design values, aesthetics, cherished heuristics, familiar patterns, and favored practices are baked into our umwelt. We are shaped by both positive and negative experiences. We've learned appropriate reactions to familiar design problems and effective ways to approach a new design. We've also learned to shy away from design solutions that seem clunky, fragile, awkward, overly simplistic or overly complicated.

To effectively apply any pattern, we must unfold it within our design context using our unique pre-existing perceptions, biases, knowledge, design aesthetics, and skills.

There is a crucial question that needs to be addressed: How should we as designers approach learning and applying new patterns? New experiences and new-to-us patterns have the potential to (re)shape our umwelt, but first, they have to be unfolded in our current environment. Not only that, ideally, the pattern should either smoothly "fit" well with what we already know and do, or eventually cause us to readjust our perceptions of what constitutes a better solution.

The first hurdle to overcome, at least that has been in our experience, is to silence our inner critic and listen to what the pattern is trying to tell us. Our preconceptions can limit our ability to see a pattern's full potential. In order to give a pattern a fair trial, we need to envision how we might unfold it and imagine its effect. Then, there is a matter of practically unfolding the pattern in a realistic design context. That might require significant practice. At first, we may miss subtle cues from our context that require adaptations to make the new pattern "work." Finally, that pattern's impact on our design may not be what we had imagined. So, we need to think about that and perhaps try out other ways of unfolding it. Or, we may decide that the pattern isn't a good fit. This sounds like a lot of work. It can be.

Unfolding a pattern's potential or an important design idea/concept, can take time. It can't be rushed. Eric Evans speaks of intentionally giving a design concept an awkward name, until he can figure out what it really should be in his emerging design. That might take days! Or weeks! We've both encountered

//this is ugly

commented code. A reminder that the unfolding is unfinished. And when we think we know enough to do a good job at it, we complete it.

With experience this gets easier.

4.3 The Conversation between the author's and designer's umwelt

Regardless of approach, any pattern author embodies aspects of their umwelt into their pattern descriptions. How they name, explain, organize, explore and motivate the pattern comes about through their direct experiences and/or observations. Patterns take on their particular point of view.

Not every pattern author is expert in the topic their patterns address. Many are. However, if they are not, they can develop expertise through practice and experimentation. Some pattern authors who lack expertise in a particular field take the alternate approach of mining the expertise of others. Their goal is to formulate patterns that embody that expertise. They uncover that expertise through direct observations and deep conversations. Finding and writing about patterns this way is another kind of unfolding. Learning about and then organizing and encoding the wisdom of experts into patterns, can result in fresh perspectives. It can draw out essential tacit knowledge that experts on their own may not be able to articulate.

One important choice any pattern author makes is how to effectively convey the potential of their patterns. Based on assumptions they make about their patterns' intended users (and their umwelts), they might include more or less detail. The challenge is to enfold enough of the essence of a pattern in its description so intended readers can unfold it in their context.

Traditionally, at PLoP conferences, in a writers' workshop we ask the author to explain the intended audience for their patterns before we discuss their work. Hopefully that puts workshop participants in the right frame of mind to read and remark on the pattern as if through the eyes of a potential pattern consumer.

Yet how easy is it for us to imagine others' umwelts? Given significant pattern reading and writing expertise and background, workshop participants can offer pretty good advice on how to improve writing and suggest ways that pattern descriptions could be made more compelling/convincing. Seasoned workshop participants with enough affinity to an author's work, can spot strengths and weaknesses in patterns that might cause the author to clarify their writing or significantly revise their thinking.

But we workshop participants may be deluding themselves. How useful are our remarks on pattern details to any author? They might take our suggestions to heart, or dismiss them entirely because they seem inappropriate. Our advice may not be a good fit with their umwelt.

Feedback received during shepherding, on the other hand, can be quite different. If a pattern author and shepherd "click", there can be significant sharing of knowledge and "tweaking" of the patterns based on feedback from the author—an "unfolding" of knowledge as better pattern descriptions emerge.

4.4 The ultimate user's umwelt

As we have seen, authors and designers using pattern descriptions have their unique umwelts. Their experiences and agency affect the situation at hand. But as patterns come into being as real-world instances, they face their own umwelt. First, there are the specific design requirements of an actual context, not the generalized context of the pattern but the actual context with all its specifics. Second, whenever we implement a solution there are more

stakeholders affected, and each of these stakeholders has her own umwelt. So, if we build something for others, or in a participatory design process with others, then each of these individuals bring their own umwelt. A pattern author can have her own certain ideas and values that make the pattern's form a solution in a given context. A designer using the pattern, who is guided by the pattern description to build that solution will necessarily add her own values and ideas. But the story doesn't end there. Everyone who is in contact with the solution will add another layer of ideas and solutions. The pattern only comes to life by being used by these individuals.

When Alexander et al. (Alexander, 1977) described the pattern PROMENADE, they captured all the forces that lead to this solution. When a city council or a community plan to create a PROMENADE they will have their own expectations on how it unfolds in their community. They add additional forces. And when the PROMENADE is built, the actual people who walk along the PROMENADE make their own experiences. Life unfolds on the PROMENADE, if and only if it fits with the umwelt of the community that uses the PROMENADE. If the PROMENADE does not resonate with the community, they will not take care of it. If they are careless and throw trash on it or neglect to maintain it, it will not come to life fully.

Therefore, we need to consider the umwelt of the pattern author, the umwelt of the designers using the pattern, and the umwelt of the community who brings the patterned solution ultimately to life.

In theory, a design situation has a number of known forces. Such things "force" a designer to make specific decisions, i.e., covering issues that drive the design process. That is why forces provide very important guidance for designers; they shine light on the issues that need consideration. But! One must not be misled that such forces provide ultimate answers. Rather, each situation is different. Forces give us an illusion of completeness and rationality—the forces are the cause-effect glue between context and solution. In practice, however, problem solving happens under the conditions of bounded rationality (Rowe, 1990). That is, we designers do not know the entire umwelt in which we will unfold a pattern. Nor can we capture the intents, motives, values and ideas of all our potential users.

So, we can have a solution that should work, but is sabotaged by certain stakeholders or nay-sayers. In such cases the designer has to adapt the unfolding process to make the pattern "work." Participatory design and co-creative processes are ways to account for the situation beyond the designer's own umwelt. These processes can help to find hidden forces that are based on expectations and demands of a community.

To better understand which forces are the primary drivers in a specific umwelt, we need to account for the individual experiences of the pattern's authors, the designers who use a pattern, and the users who will live with the solution—or rather bring the solution to life. What use is a PROMENADE that is deserted because it does not belong to the community? What use is a software pattern if the development team does not make use of its capabilities? For example, if a team does not invoke the interfaces that could decouple components, then those interfaces are of no use.

4.5 Realizing the pattern's potential

Patterns need the right ground to unfold their full potential. For example, the system needs to provide some foundation for the use of a pattern. And the designer needs to make sense of the pattern.

Patterns should lead to informed action. That is, a pattern description should not tell you what to do but what to think about to make a good decision about your next design steps.

A pattern gains meaning when it fits and responds to its environment. The term "fitness" suggests that there is a connection to the theory of evolution; good design would be considered as the survival of the "fittest" or "best of breed" (Buschmann, Henney & Schmidt, 2007). Alexander, however, claims a necessity of forms (Alexander, 1964; Alexander, 2002). This is coherent with D'Arcy Thompson's (1942) view on evolution that assumes a natural (physical) necessity based on forces.

Simon (1996, p. 196) shows that "complex systems will evolve from simple systems much more rapidly if there are stable intermediate forms than if there are not." Encapsulation is one means to handle complexity (Alexander, 1964) and evolve solutions nearly independently. A complex system should be decomposed properly into sub-problems that can be solved one at a time and that are well understood without losing the big picture of the general problem frame.

4.6 Encapsulate into patterns

Encapsulation is an important principle in software design. Large software systems can consist of thousands of classes and/or functions. If every small change would affect the whole architecture, it could not be maintained. While wholeness suggests that the parts have to fit together and orchestrates the shape of a part, the inner structure of a part is nearly independent from other parts.

With software patterns, not only is it important to encapsulate related things, but also to purposefully hide them from external view. For example, if the two variables x and y represent a Point, then these two variables form a whole and should be encapsulated, usually as a class or prototype object. However, to protect their inner centeredness—the very close relation of x and y, with x and y being more closely related to each other than to anything else—we should also hide their implementation details and control their access. Making the actual data storage private and providing methods to change the Point composed of x and y in meaningful ways makes the design robust and resilient to ill-conceived external changes. Essentially, the encapsulation of x and y into a Point makes the concept not more abstract by removing things, but by making it denser with meaning.

Such protection for encapsulated entities can also be found in physical structures. An apartment is itself an encapsulated entity. My apartment is distinct from my neighbor's. A door with a key lock controls access to this encapsulated area in which all my private belongings are closely related by proximity and the stories I can tell about them. Now, an open door means that anyone can rearrange the inner structure, add or remove things, and even create new connections to other spaces. They might even run an electric cable from one apartment to the next without my control. If I want to keep the strong centers of my apartment—both physical and fictional—I must protect it from outside invasion.

A pattern should be dense—compressed, enfolded—in a way that it captures not just an abstraction but the potential to be unfolded in many ways (a million times over) in accordance with its umwelt.

4.7 Enfold the potential in pattern descriptions

Patterns are there in the world—like elephants—but to share our knowledge about them we need to pick some sort of representation: stories, pictures, illustrations, models, songs, dances. We think that we pattern authors should be experimenting more on how to represent patterns. Perhaps we should also invite the reader to do their own experiments as they learn how to unfold our patterns. This might be in the form of design challenges, or prompts for reflection rather than ready solutions. And we should encourage readers to make variants of the recipes our patterns provide.

For this to be effective, the examples a pattern description presents should show some of the variance of the pattern's unfoldings. It often makes sense to do so using multiple representations. Examples are important for explaining patterns. They are not only evidence for the pattern's existence but have educational value. There are running examples, there are illustrative examples, and there are examples constructed solely for educational purposes.

With examples and multiple ways of representations, we invite pattern readers to build their own schemas in their own minds. By just showing one elephant or by only describing it with words, readers do not get the idea. They only learn about one instance. With multiple representations, examples, and experience reports, they can connect the pattern's idea to their prior knowledge. This empowers them to construct their own instances—either for real or in their minds or as thoughts shared in a design conversation. That is, their expressive power increases just as Christopher Alexander intended with the concept of pattern languages for design.

5. PATTERN SMELLS

An experienced pattern practitioner—someone from the patterns community or someone who has written, read or applied many patterns—intuitively gets a feeling whether a pattern seems right or wrong. But sometimes it is hard to express why the pattern does not work. Based on the problems identified in the previous section, we propose some "pattern smells" that make it easier to name what is wrong with a pattern, its description, or its application.

5.1 The smells of questionable solutions

Untested solutions: Patterns can and should be tested. The more often a pattern is a success, the more trust we can put in it. Rising (1998) highlights that "patterns are not theoretical constructs created in an ivory tower, they are artifacts that have been discovered in more than one existing system." However, since patterns are abstract entities that can only manifest in real instances, they are also theoretical constructs. Writing a pattern description is a creative and original act. Sometimes the pattern format or the thinking in patterns is used to propose a new solution that hasn't been tried before. This is not a bad thing per se, as the formulation of patterns helps to integrate and address many different forces and find balance in a complex system. The smell occurs when such a proposal is not declared as such. It would be beneficial for the pattern community to differentiate between patterns with tested or validated solutions versus speculative solutions. An untested solution never met a real umwelt.

No variance: A pattern that shows no variance is not very useful. By definition, recurrent structures are also patterns—and that includes structures that repeat themselves without variation such as fabric patterns or wallpaper. But such patterns are not interesting to designers. A designer needs to be able to design; that is, she needs to be able to shape the final form. A useful, interesting design pattern is one that lets you create the same thing a million times over without ever doing it the same way twice. You don't want to have the form of a single house, you want to know how to generate the forms of many different houses. This smell sometimes occurs if the authors do not have enough expertise in the field. Their experience is based on too few examples. To see and understand a pattern one needs to understand both its essence and its variations. Without variance the umwelt is fixed, it doesn't allow for any adaptations.

Too much variance: If there are too many differences between the examples a pattern builds on, then there is no common form. We may call it variance or "too much variance," but what it often is, it is a mix-up of different forms sharing some properties. A red apple and a red car both share REDNESS. But are they really instances of a REDNESS pattern? Such confusion often occurs when two solutions share several properties (being RED, LARGE, FAST, …) and especially if they share the same effect or goal. Two different pathways can lead to the same destination. That does not make those two ways the same pattern. Each of the two paths have their umwelt! Hence, without a clear form there is no way of assessing the fitness between this form and context.

However, if the designer follows the same trail and makes some detours on her own volition, then she is following the same pattern, yet with some variant steps. This actually makes the pattern more alive, not less.

Lack of openness: Openness refers to the flexibility and adaptability of patterns as well as their compatibility with other patterns. Openness does not only matter at the time of creating a system, but also impacts its lifetime of maintenance. In order to adapt to the specifics of a situation, a pattern is more open if it allows a variety of sub-patterns (open to the replacement of sub-parts). In addition, a pattern that does not narrow the options for how to proceed is more open (open to different next steps). A resulting context that can be addressed by more than one sub-pattern is more open. Likewise, a context that is broader, e.g., that includes more previous or next steps, makes the pattern more open for the combination with other patterns. Without sufficient openness, the pattern cannot co-evolve with other solutions and along with the user's umwelt.

5.2 The smell of poor pattern descriptions

Too abstract or too specific: To be open and allow for variance, we often need to make some generalizations. However, if the description is too abstract, no one else can apply the pattern. In order to unfold the structure of the pattern, one needs to know the critical design decisions—the essential elements that make the pattern form. On the other hand, a pattern description can be too specific. If it prescribes every single design step and every decision to choose the next step, then one can find it impossible or tedious to adapt it to their specific situation. An abstraction is a mapping of some part of the world that intentionally omits irrelevant features. However, if we choose an abstraction that loses essential information, nothing is gained by abstraction. Abstraction should be used only "in moderation" (Gabriel, 1996, p.19). Representations of patterns should illustrate the structural quality of recurrent solutions, i.e. the fundamental relations of the elements of the pattern (Alexander, 1979). In this smell, the umwelt of the pattern author and the pattern designer do not match. A pattern description that is formulated in too abstract terms assumes too much prior knowledge of the reader. Hence, the reader cannot unfold the meaning in their umwelt or they may miss important points. A too specific description is not only redundant for experienced readers but may narrow the potential actions by being overly prescriptive.

Noise that hides gaps: An experienced designer knows when to break the rules and improvise. But inexperienced designers find it much harder to distinguish between essential and optional but recommended steps. To use a path metaphor: sometimes a change of direction is needed to get you to your destination, sometimes there are different options and you can either describe all of these options (boring!), describe just one option (but how does the designer know that there are others and that the decision for one option is not essential?), or none (let the designer figure it out). The worst kind of specificity is extraneous detail. Good: "after you see the town house, turn right" Better: "immediately after you see the town house, turn right." Bad: "move straight on and pass the coffee shop, move straight on and pass the bakery next to coffee shop, after the coffee shop and bakery, you see a bookstore, still move

straight […] and then you will see the town house, there you turn right." The bad description is not wrong. It's just full of extraneous details so there is a significant risk of missing the essential direction to turn right after the townhouse. The "good" description may be lacking the necessary detail that you should turn after the townhouse. A "better" description adds that extra bit of detail in just the right places. Too much noise on the one hand, and forgetting the important bits might indicate that the author still has some confusions about her own umwelt—like not seeing potential dangers or making false assumptions.

Lack of new insights: Not everything written as a pattern is useful, especially when it simply captures the obvious. Common knowledge about things can be written as patterns but what is the point? Likewise, if you only put common knowledge into a pattern description it can be irrelevant. However, what is relevant depends on the situation of the designer. Tables are certainly relevant in the world. But is the description of a physical TABLE pattern relevant? Probably not for most people who want to design their living rooms. They know what a table is and what it does. But if you are writing the pattern TABLE for carpenters, a description of materials, good table measures, and principles of stability could be interesting. Understanding the umwelt of the designer is therefore an important consideration as you decide to write a pattern.

Isolated patterns: A pattern should not be an island. Patterns can be nested, overlapping, competing and collaborating. In pattern languages the patterns themselves introduce connections to other patterns. This is done by "related patterns" sections or referring to other patterns by their names. If the context of a pattern refers to other patterns, this indicates that its application makes sense in the context of those patterns. These descriptions tell us where to proceed if in a resulting context not all the forces are resolved: "Each pattern then, depends both on the smaller patterns it contains, and on the larger patterns within which it is contained" (Alexander, 1979, p. 312). Moreover, connections between patterns provide orientation in the pattern vocabulary. They make explicit that a "pattern uses another pattern, a pattern refines another pattern, or a pattern conflicts with another pattern" (Noble, 1998). A pattern description that stands alone, in isolation, without any referential information, lacks connection. It misses its umwelt.

5.3 The smells of wrongly applied patterns

Used in a clumsy way: When somebody selects a pattern but lacks the skills to implement and adjust it, then it is more likely that they will create a caricature of the solution. This often happens if the pattern is applied in a mechanistic way without reflecting about the design decisions. The designer needs more skills to make it happen in her particular umwelt. This is closely related to the next smell.

Used for the sake of using patterns: Patterns do not make better solutions per se. Often patterns first increase local complexity by adding more structure. They do so in order to decrease overall system complexity by making the system more flexible. But if the overall system is simple and does not require that flexibility, then the pattern merely increases the complexity with no benefits. If somebody says proudly that they have used "all the (fill in the blank: GOF, Domain Driven Design, Microservice…) patterns in their system" that is a big smell. All the patterns…that's a lot. Quantity isn't what's important. It is the overall quality that matters. One has to pick the right patterns, the patterns that fit. If somebody uses patterns without being able to explain why specific patterns were chosen, that's a smell. It's not the form, it is the form in context that makes a good solution.

No resonance, no feeling: A designer should add to the design by making additional design decisions and adjusting the pattern's structure according to their design context and personal aesthetics. She should add her own character to the design! Without resonance between designers and their solutions, we get cold and mechanistic designs that don't take any responsibility for improving their environment. This is a bad smell because it means the solution is (almost) dead. It cannot survive because it is not supported by the umwelt.

5.4 The biggest smell of all

Lack of wholeness: Wholeness emerges out of the deep interlock of solutions and their umwelts. But you can see or hallucinate solutions without wholeness, you can lose wholeness in the capturing and description, in the making, and in the using. There are lots of ways to screw things up. Ideally, any solution has to work completely and as a whole.

Every programmer is well aware that small errors or design failures can cause an entire system to break. Each part of the system has to be designed in a way that it serves the needs of the whole, giving it a gestalt (Gabriel, 1996, p.13). Wholeness also means that a system has a new quality that cannot be reduced to its individual parts. A system is more than its sub-parts. A whole form emerges out of the dynamic interplay of smaller whole forms and serves itself to a larger whole. Furthermore, the nesting of wholes implies that the designer has to mind the fitness "at several boundaries within the ensemble at once" (Alexander, 1964, p.18) and that "there is a perfect balance between the needs of the parts and the needs of the whole" (Alexander, 1975, p.14). Wholeness is indeed an important concept that can be found in all of Christopher Alexander's work. An umwelt that is not whole is broken. As designers we should be inspired to make things more whole. If we can repair small parts of the world, that's a good thing.

6. BRINGING PATTERNS TO LIFE

What is our main motivation when we write patterns? Don't we want, eventually, to make this world a better place? We care for quality in design, for solutions that matter and help society—programmers, house inhabitants, students, organizations, communities—feel more alive. Isn't it that we not only care for existing good solutions, but want to see more of these good things in the world? That is why we capture what has been done successfully to improve systems and our environment. We not only want to see and recognize good designs, but seek to understand them and provide agency to ourselves and others, helping us all realize the potential of good patterns.

To do so, we have to find existing solutions that feel alive. And we need to describe them in an engaging and illuminating way. As readers, we need to feel the potential of the pattern, it needs to resonate with our umwelt. Moreover, we need to feel empowered and inspired to make this potential realized—guided by the reflections, explanations and suggestions provided in the pattern description.

We can consider patterns as seeds that unfold with the help of designers. But if a seed is put into rotten soil, it will be very hard to grow to its full potential, even with the best seed and the best gardener or caretaker. Hence, we need to be aware of the cultural umwelt where we sow our pattern seeds.

The *Design Patterns* (GOF) (Gamma, 1994) were a success because they were planted at the right time when developers were starving for help in their object-oriented design processes. This was different for *Alexander's A Pattern Language* (APL). APL has wonderful patterns, seeds for growing lively cities and neighborhoods. But did it find the right soil? Were architects waiting for it? Or did many reject it, rightly or wrongly, as an arrogant assault to their common practice? Our argument is that for a pattern or pattern language to be successful, it is insufficient for it to have potential. It also needs to meet the umwelt in which this potential is keenly perceived, welcomed, and realized. And this means that both the designers and, later, the users have to appreciate and feel the value of the pattern.

6.1 Finding vivid patterns

So, a primary question to be addressed is what is the vividness of the patterns we perceive. Depending on our umwelt experiences, we perceive patterns differently. We have differences in recognizing forms and assessing them. I may overlook a form due to my lack of attention or lack of experience. Or, I may overlook that this form provides a good solution in some particular context but not another. I may or may not perceive that this form is a general approach that can be unfolded again and again.

To judge a solution (or even a form) as good or bad is both subjective and objective. It is subjective in that I may put different priorities on its underlying principles and design criteria than you. It is objective in as much as there are principles and criteria that I can name and explicate, and we can agree to some extent on their meaning and importance. And, of course, the pattern smells we mentioned previously are to be avoided.

Therefore, as pattern authors, share patterns that are vivid, exciting and envision a better future. Select the best of breed. Aim for quality rather than quantity. Start with the patterns where other designers and users think: I need this, I want this, I will do this, I can do this. Less is more. Do not overwhelm, but let designers feel that they can do it. Start describing the solution in a way that shows them that they, too, can put it to good use.

6.2 Vivid pattern descriptions

We need more vivid pattern descriptions! Formality and precision are important to get things right. Clarity and conciseness are key. But that doesn't mean descriptions should be boring or hard to understand. According to Whitehead "Words in general indicate useful particularities. How can they be employed to convey a sense of that general character on which all importance depends? It is one function of great literature to evoke a vivid feeling of

what lies beyond words" (Whitehead, 1938). And great pattern descriptions are great literature in our humble opinion. But this is not always enough.

Until recently, we assumed that good pattern descriptions are composed of text and include simple illustrations. But there are other perhaps more vivid ways to describe and communicate a pattern. Alexander points out that each person must have his or her own version of a language in order to make it a living language: "A living language must constantly be re-created in each person's mind" (Alexander, 1979, p. 338).

Pattern cards are very short descriptions or illustrations or abstracts. They can be used as design tools to combine multiple patterns into rich solutions. They can also be triggers to tell stories about how to use or create the patterns. Iba (2012) has developed a concept to learn a pattern language through personal communication: A large group of students gets sheets with brief pattern descriptions. The students highlight which patterns they have experienced and which they want to learn about. Students who know an example for a pattern share their story with students who do not yet know the pattern. This way, concrete cases of a pattern are discussed in personal communication rather than in written form.

You can dramatically perform and embody patterns. Even software patterns can be dramatized to show and feel the dynamics and relations of the patterns' structure (Vranić & Vranić, 2019). Mangas about patterns are another great way to involve readers in the actions and feelings of the pattern (Iba et al., 2023). Patterns of buildings or spaces can be experienced as you walk through the physical spaces that embody them. You can play and walk through software code and, more important, its execution. Photos as found in *A Pattern Language*, or 3D videos are also good representations.

A pattern description should not only grab the reader's attention, but also support the reader in making good design decisions. A pattern language provides a gateway to build things. Each pattern creates part of the overall design. One pattern leads to another. And so, pattern language authors should identify the entry patterns and potential pathways through their language.

Make it fun to use the pattern description. Let people talk about it. A pattern description should be a tool for discussing design choices and making good next steps. It should not prescribe, but invite a fruitful conversation—with the emerging design, with other designers, with fellow patterns enthusiasts.

If patterns are seeds of potential designs, then let the designers know what caretaking is involved to make their own solutions thrive.

There are certainly different skill levels and levels of freedom given in any pattern description. Consider the building of an Ikea table. Almost everyone can "make" the table according to the instructions provided. But is that an unfolding of a pattern? No. The designer of an Ikea table initially unfolded a pattern to build many identical tables. It's not the consumer who is designing an Ikea table, they are only assembling it. But the consumer still has agency. While the Ikea table has exhausted its potential and is a finished design, it can still increase the potential for a living room in the making. One can use the table to unfold their own living environment in an affordable way. Of course, the most vivid spaces are those where everything fits together and each bespoke piece of furniture is adapted and unfolded to fit into and enhance that space.

So, we have the Ikea table and bespoke table. But even bespoke tables still need to be constructed. They don't make themselves.

There is one more thing to consider: sometimes the designer cannot build the thing itself but can only ensure that the environment is right. Like a seed that needs to be nurtured but only can grow and unfold on its own. Education patterns are often like that. One can design the right teaching scenarios and learning spaces, but successful learning unfolds only through the interactions and activities of the students. Likewise, many of the patterns in *A Pattern Language* are not solutions the moment they are built, but only when they are lived. The STREET CORNER only thrives if the community uses it in lively ways.

So, the unfolding of a pattern's potential needs to have this community in mind. Ideally, pattern description should provide guidance for this, too.

6.3 Patterned implementations that are alive

The twenty-three patterns in *Design Patterns* (GoF patterns) were picked up quickly and spread fast. They are alive in many systems. Are they the best? *Patterns of Software Architecture* (POSA) (Buschmann et al., 1996) came just a little later and their great patterns had to remain in a niche because of GoF pattern dominance. The GoF patterns, on the other hand, have been replicated many times: both the actual solution implementations and their descriptions. There are variants for different programming languages and for specific domains such as game development.

Patterns evolve as they are re-created in our world. They are adapted as we learn more and more about what works and what doesn't. The original idea of *A Pattern Language* was to have a folder in which the pattern descriptions could be updated and be replaced[2]. The Wiki software, originally invented by Ward Cunningham, was intended to be a lively place to share and evolve patterns, and allow for this continuous updating. And newer books with rewritten GoF and POSA patterns have taken changes in software development environments and tools into account.

Both the pattern implementations and pattern descriptions can be alive. A pattern implementation is alive when it is used and the "Quality Without A Name" emerges. When people are dancing in the street because the neighborhood is built that way. When developers use a STRATEGY (software) pattern not only to plan a system, but to build, maintain and extend it vividly.

7. CONCLUSION

A pattern description is alive when it is updated and captures the everchanging design space of that pattern. The patterns of an OBSERVER, of a STREET CORNER, a PROMENADE of a MAKER SPACE are the coherent gestalts of all the instances. If there are new instances, the pattern changes accordingly. What a CAR is today is different from what a CAR was a hundred years ago.

To keep the patterns alive (both implementations and descriptions), one must not forget necessary care and ongoing maintenance. To implement a pattern in software code is not necessarily a good thing forever; the software and its patterns may need to be adapted in order for the software to continue to grow and evolve. Likewise, if pattern descriptions stick to a reality and umwelt that is not there anymore, they become meaningless and, in the end, dead.

Therefore,

- Care for the patterns you choose. Think about the umwelt they will be embodied in.
- Care for high quality and useful descriptions. Think of the designer's umwelt.
- Care for patterns and their maintenance. Because the umwelt always evolves.

[2] Rebecca made the suggestion to Addison-Wesley in her pre-publication review of Design Patterns that they consider publishing the book in a loose-leafed folder so that changes and additions could be offered to purchasers on a regular basis. This suggestion wasn't because she deemed them incomplete, but rather, that naturally through use she imagined that they would evolve and be extended.

REFERENCES

Alexander, C. (1964). *Notes on the synthesis of form*. Cambridge: Harvard University Press.

Alexander, C. (1975). *The Oregon Experiment*. New York: Oxford University Press.

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

Alexander, C. (2002a). *The Nature of Order, Book 1. The phenomenon of life*. Berkeley, Calif: Center for Environmental Structure.

Alexander, C. (2002b). *The Nature of Order, Book 2. The phenomenon of life*. Berkeley, Calif: Center for Environmental Structure.

Alexander, C., Ishikawa, S., & Silverstein, M. (1968). A pattern language which generates multi-service centers. University of California, Berkeley: Center for environmental structure.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). *A pattern language*. New York, USA: Oxford University Press.

Appleton, B. (2000). Patterns and Software: Essential Concepts and Terminology.

Ball, P. (2009). *Shapes - Nature's patterns: A tapestry in three parts*. Oxford: Oxford University Press.

Buschmann, F., Henney, K., & Schmidt, D.C. (2007). *Pattern-oriented software architecture. Volume 5: On patterns and Pattern Languages*. West Sussex: John Wiley & Sons.

Coplien, J. O. (1996). *Software Patterns*. New York: SIGS Books & Multimedia

Coplien, J.O. (1998c). A Generative Development – Process Pattern Language. In L. Rising (Ed.), *The Pattern Handbook* (pp.243-300). Cambridge: Cambridge University Press.

Coplien, J. O. (2004). The Culture of Patterns. Computer Science and Information Systems. 1(2).

Foote, B. (1997) In R.C. Martin, D. Riehle, & F. Buschmann (Eds.), *Pattern Languages of Program Design 3*. Reading: Addisson Wesley.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

Gabriel, R. P. (1996). *Patterns of software: Tales from the software community*. New York: Oxford University Press.

Goldstein, E. Bruce. (2009). *Sensation and Perception*. Wadsworth Pub.

Iba, T. (2012). A Pattern Language for Designing Workshop to Introduce a Pattern Language. 18th European Pattern Languages of Programs conference. Writers' Workshop version. Irsee, Bavaria.

Iba, T., Tanaka H., Adachi, S., Ota, M. and Tajima, U. (2023). Pattern Manga: Attractively Expressing Patterns of a Pattern Language in Manga Style. In Proceedings of the 30th Conference on Pattern Languages of Programs (PLoP '23). The Hillside Group, USA, Article 10, 1–42.

Karlsson, E. Everything that turned out well in my life followed the same design process. (2024) Escaping Flatland blog. Available at: https://www.henrikkarlsson.xyz/p/unfolding (Accessed: 28 August, 2025)

Kohls, C., & Scheiter, K. (2008). The relation between design patterns and schema theory. Proceedings of the 2008 conference on Pattern Languages of Programs (PLoP). Nashville, Tennessee: ACM.

Lea, D. (1994). Christopher Alexander: An Introduction for Object-Oriented Designers.

Majetschak, S. (2007). Ästhetik zur Einführung. Hamburg: Junius.

Noble, J. (1998). Classifying relationships between object-oriented design patters. Australian Software Engineering Conference (ASWEC), pp. 98-107.

Quillien, J., (2008). *Delight's muse on Christopher Alexander's The nature of order: A summary and personal interpretation*. Ames, Iowa: Culicidae Architectural Press.

Quillien, J., Rostal, P., West, D. (2009). Agile, Anthropology, and Alexander's Architecture: An Essay in Three Voices. OOPSLA 2009. Orlando, FL, USA.

Rising, L. (1998). *The Pattern Handbook*. Cambridge: Cambridge University Press

Rowe, P.G. (1990). *Design Thinking*. Cambridge: MIT Press.

Simon, H.A. (1962). The architecture of complexity. Proceedings of the American Philosophical Society. 74. 29-39.

Simon, H. A. (1996). *The science of the artificial*. 3rd edition. Cambridge: M.I.T. Press.

Thompson, D. A. W. (1942). *On growth and form*. Cambridge: Cambridge University Press.

Vranić, V. & Vranić, A. (2019). Drama patterns: extracting and reusing the essence of drama. In Proceedings of the 24th European Conference on Pattern Languages of Programs (EuroPLop '19). Association for Computing Machinery, New York, NY, USA, Article 4, 1–9. https://doi.org/10.1145/3361149.3361153

Weiss, M. (in print). Patterns as Distinctions. Springer: EuropPLoP 2025 proceedings.

Wertheimer, M. (1938). *Laws of organization in perceptual forms*. London: Harcourt, Brace, and Jovanovitch.

Wirfs-Brock, R., Wirfs-Brock, A., & Wirfs-Brock, J. (2024) Discovering Your Software Umwelt. In Onward! 2024: Proceedings of the 2024 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. (pp. 288-302).

Wirfs-Brock, R. (2010). The Nature of Order: Inspiration or Esoteric Distraction? Keynote at PLoP 2010. Reno, Nevada.

Wirfs-Brock, R., & Kohls, C. (2019). Elephants, patterns, and heuristics. In Proceedings of the 26th Conference on Pattern Languages of Programs (pp. 1-15).