Elephants, Patterns, and Heuristics

Rebecca Wirfs-Brock and Christian Kohls

Abstract
This essay discusses the challenge we face when we try to capture the wholeness of design solutions. It is argued that patterns are observable phenomena of solutions. To represent these phenomena, a pattern author needs to generalize and omit information. Heuristics play a central role to fold and unfold information in the design process. Thus, they can explain how an experienced designer can generalize from existing solutions; and they can explain how an experienced designer can unfold and generate new solutions based on patterns. As this folding and unfolding of information and knowledge seems to be quite an abstract concept, we have chosen to make our point by discussing elephants. Like patterns, elephants are an observable phenomenon, a pattern in nature. Many different descriptions, representations and accounts of elephants exist. Many people would claim to know what an elephant is while they have actually only little or limited knowledge about elephants. This analogy helps to understand how at the same time we know and do not know what a thing is.

1. Introduction

Patterns are recurrent phenomena that can be found in all domains of design. We can directly observe and experience these phenomena both as designers and users. However, very often it is very difficult to find, generalize, explain and understand these phenomena. The literature genre of pattern descriptions tries to capture patterns by not only discussing the solution form. Context, problem, forces, solution and consequences are required perspectives that should be taken on each pattern. This holistic analysis is based on Christopher Alexander's design theory, as outlined in A Pattern Language [AISJFA77], The Timeless Way of Building [Alex79], and other works.

Yet little discussion takes place about the challenges all authors and pattern researchers face when they try to preserve the wholeness of the observed solution phenomena. As each pattern is a generalization, authors have to skip details. They have to pack their available information in a way that it is useful to other designers. And other designers, to use particular patterns need to be able to unfold new solutions based on both on the pattern descriptions and an understanding of their design context. This is where personal design heuristics are required. Patterns do not provide all details because each situation is different. To unfold a specific solution from a general solution requires implicit knowledge, design experience, and the ability to make a series of tactical design decisions based on personal design heuristics.

The level of detail a pattern descriptions provides depends on the target audience. Experts have a huge set of personal design heuristics. Hence, they are capable of unfolding new solutions from general descriptions. Novices, on the other hand, need much more guidance. Thus, pattern descriptions written for them need to be very specific – up to an extent where they describe example solutions rather than a generative pattern.

The information we have and the information we provide about patterns is therefore a critical factor. Likewise, the representation of information and our research tools for gathering this information are critical.

And this is where elephants enter the stage. Like patterns, elephants are phenomena in the world. We can observe them, we think we know a good deal about them, and there are many different ways to describe and represent elephants. Yet there are many misunderstandings, myths and missing information about elephants!

If we understand the challenges (and solutions) in capturing, communicating and using knowledge about elephants, we can get some insights about how to better capture and share patterns. Because elephants are a pattern in nature. And everything we can say about the science on elephants, we can say about research on patterns. Thus, in this paper we want to talk a lot about elephants.

We will first discuss the "elephants in the room" in the pattern community. That is we will address common challenges this community is aware of but often ignores. Then we will use the phenomena of elephants to illustrate how descriptions and representations can be used to share knowledge in different ways. We will draw analogies between accounts about elephants and accounts on patterns.

As we have to reduce the information when we describe complex phenomena (both in nature and design), we discuss what we learn from asking the right questions and how heuristics fill the knowledge gap.

Finally, we will discuss how stories are often used to distill the wholeness of a phenomenon into a textual form. However, we will also see how stories can be misleading and build myths.

2. The Elephant in the room

Various meanings of "elephant in the room" are:

- An obvious problem
- An inconvenient truth
- Something that is taboo
- Something deliberately ignored
- Something that is obvious (so obvious to everyone that we don't mention it)

We believe that the elephant in our PLoP room has all these meanings. *Nobody is talking about the elephant in the room.* Or if they are, they aren't speaking very loudly.

We hope to address some of these meanings of elephant in the room and offer some paths for patterns authors, educators, and software developers to explore.

2.1 Let's talk about the elephant in the room

Let's start out by stating an obvious problem: The way experts absorb information differs from novices. Historically, software patterns were written for practitioners and not novice developers. Ironically, it is the GOF patterns, written by experts for experienced developers that were and still are taught to novices. If we want software patterns to have a lasting impact, they need to appeal to a broader audience. There needs to some way that a broader, more useful number of software patterns are coherently organized and easily found.

Here's another obvious problem: Researchers find that experts, while they may not agree with each other, are logically self-consistent in their individual opinions. Patterns written by single individuals or a tight-knit group of collaborators are cohesive. Patterns from different corners are not. Experts can absorb differences of style and substance with some effort and even reconcile conflicting patterns' advice; this is a much harder task for those new to patterns or to software design.

An inconvenient truth: We mostly write patterns for people like ourselves or whom we think are like us. This limits our patterns' reach and impact.

Something that is taboo: The body of software patterns over time has become disorganized, stale, outdated, and largely irrelevant to many software developers. While new patterns are being written, they are largely ignored unless they are about a popular trending technology (e.g. microservices, event-sourced architectures). When these new patterns are published they aren't located within the overall pre-existing software pattern landscape. Consequently, they are disconnected from prior work. There is no overall coherence to the large body of software patterns. Furthermore, there's a wealth of useful, specific, concrete design advice being written and communicated to broad audiences that are not written as patterns. If we want to encourage pattern literacy, relevancy, and have a long-term impact, something needs to change.

Another taboo: The word pattern may be the wrong word for what we write. Most people (outside of software) think of patterns as being something different than what we create when we write our software patterns.

Merriam Webster lists several meanings for pattern. The first definition is "a form or model proposed for imitation**, an exemplar." The second definition is "**something designed or used as a model for making things." **While our software patterns**

often provide simple solutions, are they exemplars or models for makers? Or are they something less, e.g. gists or essences?

Something deliberately ignored: Different audiences for software design patterns need different ways to absorb, comprehend, and understand how to apply them.

Something obvious: Simply reading patterns and learning pattern names doesn't ensure that the reader can apply them appropriately (if at all). Moreover, learning about patterns may be counterproductive to learning how to exercise design judgment.

2.2 Elephants are under stress

"Elephant populations in India and also in the whole of Asia are under severe stress. The captive ones are rendered jobless due to changes in the mode of transport and lifestyle of people. The ones in the wild are also no better off, as the forests are shrinking." —Mark Shand

Are software patterns under stress? Most certainly. While captive (written patterns) may have initially have been useful, over time many have been rendered irrelevant due to changes in technology and in the way we design and build software[1]. In general, people, don't read about patterns in books in order to learn how to design. They may find design patterns in online sources or be exposed to the idea of patterns at school, but they find design advice where they will (and don't expect to read patterns in order to learn about how to design software).

Patterns written in the wild (those which have not been reviewed at PLoP writers' workshops) may be no better off than the pattern community's carefully shepherded, tame patterns. While wild patterns may be found by subcultures of developers. Regardless of their origins, both kinds of patterns are often not maintained in a sustainable way that allows for their growth and evolution based on feedback from a community of users.

2.3 Thesis

As we have seen, patterns and elephants share a lot in common. In fact, elephants are natural instances of patterns in our world. Each specific elephant exemplifies the general pattern of ELEPHANT. However, as we shall see, even for such a common category there are a lot of challenges to getting a sound picture of what elephants are and how to treat them. If we face such challenges for elephants, how can we believe that pattern mining for the dynamic field of software development is easier?

---

[1] One exception of note, that we are aware of, is the pattern collection from *Domain-Driven Design* [Evan], where the initial design patterns written by Eric Evans have been embraced and extended by a domain driven design community. These early patterns have been refreshed with additional patterns and heuristics for modeling a domain as well as designing software systems using event-sourced architectures.

Our discussion of elephants can show some obvious shortcomings about any attempt to get an objective account about things in the world and tell their story. However, we will also see that despite all these challenges, we all share a reasonably good understanding of what elephants are, and that indeed, there are good stories about elephants.

Everything we consider about elephants in the next sections can be said about patterns, and more generally, to wider-ranging software design heuristics not documented as patterns, as well. We will reflect about different ways to depict and describe elephants. We will discuss representations in general and the amount of information that is given or missing. We will learn about myths, and expert and novice knowledge. We will see why stories help to communicate. And we will draw some practical implications from this discourse at the end of this paper.

3. Sharing expertise

If we want to share our expertise, we need to communicate our knowledge about the phenomenon in some way. This is easier said than done. We all (think to) know what an elephant is. But can we describe in a few words the nature of an elephant?

3.1 Describing an elephant

Ask different people, and they will give you different descriptions – yet all these descriptions are based on the same animals we label as elephants. You can describe an elephant with a few words, a sentence, a paragraph, a full page or even in a book. The German Wikipedia article has 50,000 characters; the English Wikipedia article has 100,000 characters. Yet both articles write about the same phenomenon. And there are books about elephants that number in the several hundreds of pages.

Likewise, a software pattern can be described on a single page—sometimes even just a single card—or on several pages[2]*.* It is uncommon, however, to find books that cover a single pattern. For some social patterns, such as pedagogical patterns, sometimes there are books that discuss one pattern in full length. For example, the pattern of a learning portfolio or the pattern of brainstorming have been described in typical pattern style as well as in dedicated books [Koh14].

However, even if you go into lengthy details, each description of elephants is incomplete. One of the reasons for this is that there are many implicit structures, which are very difficult to describe without seeing or even sensing on multiple channels a real elephant.

[2] And, in the case of software patterns, not only are pattern authors challenged to simply describe the observed phenomenon; they also attempt to describe how to create a reasonable representation of that phenomenon, e.g. a stylized or exemplary design for that software pattern.

3.2 Depicting an elephant

Most descriptions of elephants are supported by visual representations. An image implicitly shows details that verbal descriptions can hardly cover. For example, the spatial relationship of nose, eyes, ivory teeth, are shown as well as their relative sizes.

If we see a photograph of an elephant that should represent the species of elephants in general, do we see an example of an elephant or do we see the pattern of elephants? The answer is both. Obviously a photograph always depicts a specific exemplar of an elephant, an exemplification of the general pattern. However, we also see the pattern itself. A pattern manifests itself in each of its instances. Hence each instance shows the pattern.

Sometimes it is hard to discriminate this pattern, because real things in the world are often manifestations of many overlapping patterns. It can be difficult to distinguish the essence of the pattern and untangle it from the surrounding context. In software design, we often see classes that participate in different patterns with different roles in each of these patterns [BHS]. Pattern descriptions, however, focus on one particular pattern. Therefore, the essential structure of a single pattern is isolated and an abstract representation is chosen.

Instead of showing a specific exemplar of an elephant, we can show a drawing of an elephant. Unique features of a specific exemplar, such as shades of skin color or variation in ear size, may be left out in such an abstract representation. If we can clearly perceive an elephant, we have preserved its essential structure. In this case, the abstract representation still manifests the pattern. Of course, the drawing is not an elephant itself, but photographs are also not the elephant. Representations that are too abstract, however, do not depict the pattern anymore: if you draw a box to represent an elephant, then the pattern is no longer discernible.[3]

An alternative to an abstraction is to use a model. A model is a good and valid instance that does not obfuscate the core structure or the essence of a pattern [Good]. For example, to represent an elephant, a picture (or 3D model) should show an archetypical exemplar of an elephant.

Likewise, the class diagrams in the documentation of software patterns are models rather than abstractions, because the more universal structure of a pattern is manifested in the concrete class diagram of a pattern. For example, the class diagram of the OBSERVER pattern has the same structural quality as a class diagram of an actual implementation; the details change—such as method names, parameter

[3] If you think to label that box, "Elephant," then you are introducing a more abstract way to represent an elephant; one without an obvious correspondence to its physical manifestation.

types, number of methods etc.—but the core structure is preserved.[4] The interactions between the pattern elements are also universal between all instances of a proper OBSERVER.

3.3 Representations and our perceptions of them

Whether you describe or depict an elephant, each representation remains incomplete. Rudolph Arnheim, in *Visual Thinking,* explains three different *attitudes* toward perceiving objects. These perceptual attitudes also apply to seeing elephants and patterns. For purposes of this essay we will call these ways of perceiving "contextually muddled," "contextually isolated," and "contextually integrated."

A *contextually muddled perception* is when an observer "perceives the contribution of the context as an attribute of the object itself. … [The observer] sees, more or less, what a photographic camera records, either because he stares restrictively and unintelligently at a particular target or because he makes a deliberate effort to ignore the context and to concentrate on the local effect." When the context changes, the object is observed as changing its character as well.

We might observe an elephant photographed in a grassy savannah and that same elephant partially obscured by trees. When this context changes, the elephant is perceived as changing its character as well.

But contextually muddled perceptions aren't just made by naïve viewers. Realistic painters are trained to practice this form of visual reduction, purposefully narrowing their focus in order to see how a given color value as it would look through a narrow peep hole, or the size and shape of an object as though it were flattened into two-dimensional plane. This observation technique is difficult to learn and requires practice.

Most photographs of elephants show the front of elephant. But what about the back? What about a view from the top or bottom?

In the second way of perceiving—*contextually isolated*—the influence of the context is purposefully "peeled off in order to observe the object in its pure, unimpaired state." The resulting object is constant, except for whatever changes the object initiates itself. Arnheim calls this a scientific way of viewing that "seeks to establish the nature of any phenomenon in itself in order to distinguish it in each practical case from the conditions surrounding it."
We could observe an elephant photographed from different angles, removing it from its context, and thus piece together a more complete, yet isolated, depiction of that elephant.

---

[4] And because the context is software design, creating a box and labeling it "Elephant" in a class diagram should not be confused with an abstraction of a physical elephant, but instead be interpreted as representation of a software element (a class) that is part of the software design.

Software design pattern solutions are commonly presented as being contextually isolated. We see a static view of the structure of the solution in a class diagram and sometimes, if the interactions between pattern elements are of interest, be shown a dynamic view of those objects of the pattern interacting in a stylized sequence diagram. Even though a richer context where the pattern might be applied may have previously been explained in text, the pattern solution itself is contextually isolated. We don't observe the solution embedded in a rich or realistic software context.

How do elephants walk? How do they behave in specific situations? Video footage can cover more of these questions, it can even record sound. Showing a phenomenon in action is critical to understanding how it behaves in the world.

Contextually integrated perception, according to Arnheim, does not attempt to eliminate the effect of the setting on the object. Instead it fully "appreciates and enjoys the infinite and often profound and puzzling changes the object undergoes as it moves from situation to situation." Arnheim claims that, "the enlightenment one gains from such varying exposure goes beyond aesthetic." Observing an object or an elephant or a software pattern in novel situations often reveals fresh information.

Even so, any visual depiction, even showing an object over a period of time, is still incomplete. Some phenomena such as smell or heat are not represented. Such information could be supplied verbally. Visual images do not show many details about the elephant's environment and social context. What about its relation to other elephants? How does an elephant integrate into its herd? What is its relation to other animals?

Likewise, we can ask for software patterns what other qualities do they have and how they relate to other patterns. This is often explained briefly in dedicated description fields in various pattern forms. But rarely do we see descriptions of the actual interplay of a pattern with real software environments.

Takeaways

The patterns we observe in a software design are much richer than any documentation or visual depiction could ever convey.

A pattern description is just one view on the phenomenon.

Software pattern solutions are depicted in isolation from any realistic context.

A pattern can be represented only indirectly; a pattern is the emergent wholeness that is common to all its exemplars. It cannot be found in one single example, however it is manifested in each example.

A good pattern description includes a model as one representative instance of the pattern.

4. What we learn

What we can know about elephants depends on the questions we ask and our observations.

4.1 Information about elephants

The answer to each conceivable question we can ask about elephants is information about elephants. The answer is contained already *in the formation* of elephants, one only has to ask the question and find ways to capture the answer. For example, if you want to know whether elephants sleep at 3 a.m., you have to observe elephants at that time of the day. If you want to know the weight of an elephant, you have to put them on a scale. Both observations (if properly done) will not alter the nature of elephants—the facts about elephants have not changed. However, once we capture the answers, we have more information about the elephants.

Hence, each pattern contains an abundance of information. Which information we capture depends on the questions we ask [Baey]. The description format of patterns requires some information to be explicated. Each pattern, and each of its actual implementations, has always more implicated information than captured. However, the description format directs to different questions and answers. Context, problem, forces, solution and consequences each ask different questions. Very often there are more detailed questions, such as what are known uses, what are implementation details, which roles can be identified? The more questions we ask, the more we learn about the solution. One drawback is that there might be fields where the answer is unknown.

4.2 Information about environments (contexts)

We may understand what elephants are but do we understand in which environments they thrive? Can we enumerate or even generalize in which contexts they fit in? Very often we see elephants in specific contexts: in films, in a zoo, in a circus, on a safari at a specific time of the year. Can we really learn from these snippets which environments elephants belong to?

Designers face the same challenge when they only observe patterns in a limited field of contexts. For example, if a designer has used a solution frequently in specific situations, the context can easily be described. However, the designer may not be aware of other situations where the pattern is likely to work well. In this case the pattern as described is over constrained. Or, the designer may not be aware of other situations where the pattern is likely to fail or for that matter, other suitable alternatives.[5] In both these cases, a description field "Contraindications" might trigger unjustified speculation

4.3 Information about causes of form (problem and forces)

Investigating the problem and forces can be compared to researching what has caused the specific form of elephants. What is their role within the ecosystem, how do they balance nature? How would their ecosystem destabilize if they disappear (problem!) and how does the specific organism of elephants fit to the environment (forces!).

It is important to understand that elephants do have already a specific role and purpose within in their ecosystem even if science still has a lack of knowledge or misunderstanding about this. So we can have fully "functioning" elephants without us understanding all the details. We can still do further research how elephants evolved and how they interact with their environment.

Likewise, we may further investigate problems and forces of existing patterns. We can identify a working solution for a pattern without being able to explain all of its causes and effects. The problem and forces sections ask why-questions. Why do we use a specific form for a solution? A force explains the cause for a specific design decision by giving the "because" to the "why." [Koh12] But sometimes we know that something is working without being able to tell why. So should we never write patterns before we have investigated all of the forces? Or should we just continue to search for more forces, better explanations, and deeper understanding of problems?

[5] However, we views patterns as simply a particularly informative form for describing design heuristics. [Wirf17] Consequently, we are reminded that "[a heuristic] provides a plausible aid or direction in the solution of a problem but is in the final analysis unjustified, incapable of justification, and potentially fallible." [Koen]

4.4 Information about the form (solution)

As we have seen, there are many different ways to describe and represent elephants, each varying in method and detail. Science has established many different forms of representing phenomena of nature. On a walk through the Natural History Museum in London, Chris found a section that curated different methods of science to capture information and knowledge about animals, including observation, recording, mapping and modelling. At the one hand we can zoom in to see more and more details. On the other hand we can generalize and leave out information to focus on the core of a form.

A solution may consist of the core solution (the thing), implementation details (the process), and things to take care of (liabilities). When we describe the solution of a pattern we also ask, what is the general structure of the solution? We can also zoom in and discuss specific details such as how to generate or implement the solution, what variations exist, where do I have to be careful. We can also explore our understanding of the benefits, costs, drawbacks, trade-offs, and liabilities of the solution. Thus, we often ask explicitly about the **consequences** of a solution.

4.5 Missing information

The more detailed we structure a pattern description, the more information we have to provide. Sometimes as we write these descriptions, our knowledge gap becomes visible. We may not understand the whole of a context yet—even if we have successfully applied a solution.

These general questions can also be misleading. For example, on which level do we discuss problems? Many software patterns address the problem of implementing a specific design. Yes, it is difficult to implement an Observer. So, the implementation is difficult and it is a good approach to present a reasonable solution. But there is a deeper understanding that a designer needs in order to "know" that pattern. What problem does an Observer actually resolve? We are not (only) interested in how to solve the problem of implementing an Observer. We are even more interested in which design problem an Observer solves. Hence, a good pattern solution should not only ask "How to do X?" in a problem statement. This "how to" is an implicit problem that must be addressed by each pattern anyways! This is because each pattern should be generative and describe how to create a solution. But a pattern should also describe what this solution is and which actual problem is solved and answer the question why we need a specific pattern in the first place.

4.6 Relevant information

There are infinite questions we can ask about any elephant. Hence, there is infinite information we can gather about elephants.

To demonstrate the infinity of information about elephants, let us consider an example. A food designer wants to test two new elephant foods. The designer invents two new products. Now he wants to know whether elephants like A or B better. If we test this, we get the answer. The information is in the elephants already.

But only by asking the question and observing their behavior do we get the answer and access to the information.

As there are an infinite number of potential new foods, there is an infinite number of questions we can ask about elephants. We can ask any kind of strange questions: Do elephants like to watch baseball? The answer is probably no, but you never know without testing!

We can get information even if it is not relevant to us. Whether A or B is more yummy for elephants may be of interest for a food designer or for the zoo management. However, most of us are more interested in general facts such as the amount of food or whether elephants eat meat.

Researchers and pattern authors try to hone in on the "right" (or appropriate) information. Yet it is important to understand that the pattern of an elephant is so much richer than any representation can ever be. And that specific information may only be relevant in certain contexts.

4.7 Superficial information

We have seen that there is an infinite amount of information we can gather about elephants. Experts have access to a significant and relevant subset of this information. Many novices, however, think they have full information about a phenomenon when they repeatedly observe only superficial information.

Many people say they know what elephants are. But do they? Do they know their weight? How many siblings they can breed and feed? **Do they know how to react if they face an elephant in the wild?**

We see the same with many software design patterns. Just think about the Model View Controller (MVC) pattern. Many developers learn about this pattern early in their education. There are more students who (think to) know what MVC before they know what patterns are.

However, most developers reduce the MVC pattern to simply a concept that separates the model from the views and controllers, thus making the code structure more organized and the development of each design element more independent. Nothing wrong with that. However, we see that even supposedly expert designers of frameworks implement MVC in many different ways that often violate core design principles. For example, model, views and controllers are separated into different folders in the source code, but still have many dependencies. And some (student) developers claim to follow the MVC architecture. But if you ask them how to add views or change existing views without changing the model they cannot provide proper answers.

Their superficial observation is that MVC separates the elements into different folders so developers can change the files independently. They miss the important

design principle of loose coupling between objects and necessary abstractions. They also fail to recognize the Observer pattern as a mechanism to notify the views about any changes in the model. Students (think to) know what the MVC pattern is without knowing and understanding the Observer mechanism, which is used to achieve loose coupling between models and views.

This kind of superficial understanding of patterns is like saying: "I was in the zoo last week and watched the elephants for a day. Believe me, now I know all about elephants."

Seeing a pattern in action does not make us to experts about the pattern. We need to get a deeper understanding. In order to become expert, we need to understand the design principles and values that led to that pattern.

4.8 Some differences between experts and novices

Both experts and novices need to remember patterns to be able to explain them to others and to use them as shorthand for design ideas.

Novices should not be exposed to the same material that experts consume. So the question is, then: how might patterns best be introduced to novices?

"We habitually observe by the method of difference. Sometimes we see an elephant, and sometimes we do not. The result is that an elephant, when present, is noticed. Facility of observation depends on the fact that the object observed is important when present, and sometimes is absent." —Alfred North Whitehead, *Process and Reality* [Whit]

We have to have contrast before we can see. And we have to see that form consistently over time.

If novices are also students, then one plausible pedagogical approach might be to introduce them to a particulate design problem. Then show them code that solves that problem that uses a specific pattern. After seeing that code and understanding what it does, then and only then explain the pattern to them.

To understand a particular pattern's importance, they might need to see this pattern as it is applied in various different programming languages and/or technologies. And then, they must be given problems that they can solve by applying that pattern.

But somehow, in order to not get stuck with the notion that patterns are strictly applied in a particular way, students of patterns need to see various solutions to a design problem and learn that there are many alternative ways to structure a

solution even applying (or trying to apply) the same pattern.[6] Arnheim argues that the kind of concept created by contextually integrated viewing is best suited for productive thinking. And we speculate that this also holds for applying a software pattern in a designer's particular software context.

Experts also like to know when to use a pattern and how it compares with other alternatives. There are always competing heuristics and multiple ways to solve any particular design problem.

Experts also like to get down into the details: When things work, what complexities there are, and what to expect during implementation. They also need to locate patterns unfamiliar to them in multiple ways (perhaps novices do so, too), but tagging patterns so there could be a multi-faceted way to search for them could be useful.

Practitioners want to solve a current problem at hand. Maybe they could benefit by patterns organized by "how to do x." But most of the time, what they are seeking is at a different level (e.g. specific coding details) than most software patterns are written. When they are searching, they are being very concrete about what information they are looking for. Pattern descriptions, however, are abstractions.

So the question is whether they are seeking an "exemplar" to copy and modify (e.g. taking code snippets from blog posts or stack overflow), or whether they also might want to know the why behind the what to do. Arnheim remarks that a concept from which everything is subtracted but its invariants facilitates definition, classification, learning and use of that learning because "[t]he object looks the same, every time it is met." [Arn] But this stripped down, essential depiction, also leaves the person lacking any concrete, realistic, tangible experience to draw upon. The rigidity of such constancy can blind the observer to any revelations offered by her particular context and prevent her from reacting in a manner appropriate to the current situation. This could lead to a clumsy misapplication of the pattern.

Which leads us to consider whether personal software design heuristics (not patterns) are another useful tool for sharing knowledge among that could enrich our pattern knowledge [Wirf18]. The unique situations in which a pattern is applied lead designers to make choices based upon their specific context and their personal preferences and design heuristics. Applying heuristics helps a designer adapt a particular pattern to her specific situation. Knowing about others' design approaches could broaden a designer's options.

Most practitioners, however, want concrete advice, not principles or lofty heuristics when they are in the thick of solving a particular design problem. But there are

---

[6] This is something, I, Rebecca used to do when teaching object design classes. I would show students several different solutions that the students themselves would come up with.[6] To do this, they had to solve a non-trivial problem.

times when they do take a breath. Maybe then they might take time to pause to reflect on what they are doing, and what worked out well and what others have tried.

As Whitehead observes: “The true method of discovery is like the flight of an aeroplane. It starts from the ground of particular observations; it makes a flight in the thin air of imaginative generalization; and again it lands for the renewed observation rendered acute by rational interpretation.” [Whit]

And that is where distillation of personal heuristic gists might be useful to more experienced designers. Rather than reading and learning others’ patterns and other’s detailed design heuristics, thoughtful developers might get in the habit of recording their own heuristics in a design diary.

Besides showing a good canonical implementation that applies a pattern, for some patterns might benefit from “how to not do it” code examples. There might also be value in recording many more specific and detailed heuristics for what needs to be considered (that is, personal, contextualized heuristics) for particular applications of these patterns, along with specific, concrete examples..

Beginners need to learn that there is no one “right way” to apply a particular pattern. They also need to experience variations in implementation of a particular pattern. To become proficient at applying a pattern to solve a non-textbook or classroom problem, they need to do more than a cut-and-paste reuse of a pattern. Not only that, but they need to be exposed to other equally valid design solutions to their problem. In order to learn to exercise the kind of design judgment, they need to be able to see and appreciate the nuances of different design approaches. This takes practice, experimentation, and acute observation.

Seeing concrete examples may not be enough to comprehend a general abstraction. Designers may need to observe slight design variances that can still be called some particular pattern so that they come to know both what this pattern means and what it means to not be apply that pattern.

As patterns writers, we too need to stretch our imaginations and envision the boundaries and true shape of our patterns through mentally exercising them. Again, from Whitehead: “The reason for the success of this method of imaginative rationalization is that, when the method of difference fails, factors which are constantly present may yet be observed under the influence of imaginative thought. Such thought supplies the differences, which the direct observation lacks. It can even play with inconsistency and can thus throw light on the consistent, and persistent, elements in experience by comparison with what in imagination is inconsistent with them. This negative judgment is the peak of mentality.” [Whit]

Takeaways

The standard description format of patterns helps us to ask the right questions about a good design. However, there might be other ways to describe the phenomenon.

Describing the forces and consequences helps us to understand how and why a pattern works. This is different from observing superficial features. Cause and effect are given. Such claims are subject to empirical evidence of falsification.

Only considering the superficial properties of a pattern is a dangerous path because developers do not understand the consequences of their design decisions.

The information provided in a pattern description needs to fit prior knowledge and preferences of the target audience.

5. Myths and stories

A little expertise can be dangerous. Thinking oneself an expert can be dangerous. If your assumptions about elephants are based on TV shows, zoos, and circus visits you may have constructed a wrong mental picture. If you face an elephant in the wild you may react all wrong. You may provoke the elephant in spite of its friendly nature.

5.1 Myths about elephants

Let us assume somebody wants to become an expert on elephants and studies them for a day in the zoo. She observes that elephants get their food at 9 a.m. in the morning from the zookeeper. So she claims that "Believe me, elephants get their food at 9 a.m. from zookeepers. That's the nature of elephants." And if you raise some doubts, our elephant expert says: "Wait a few weeks and I will provide the evidence." She then observes elephants for the next 30 days in the zoo. And guess what: the elephants always get their food at 9 a.m. More precisely, the observations have shown some variations. Sometimes elephants get the food at 9:05 or even at 9:10, sometimes even at 8:58 a.m. So, our expert adjusts the statement and claims that elephants get their food between 8:58 and 9:10 a.m. She provides a lot of evidence based on 30 days of observation. Doing this over a full year, this data becomes statistically sound. What's missing?

The answer is: our expert has ignored the specific context. The claim that elephants eat at 9:00 a.m. on every day is only valid for the specific zoo she visits. If she visits other zoos, she will learn about further variations. If she visits the same zoo 2 years later she may also see new data if the zoo management has shifted feeding times. And if our elephant expert would bother to observe elephants in the wild—in their

most relevant context—she would make quite different statements about food habits.

Misinterpreting the scope of a context is common mistake. We observe a pattern within one specific environment: one company, one programming language, or one developer team, and assume that the qualities of the pattern are true for many other contexts. However, without observing the pattern in these other contexts, we cannot make conclusive statements about these other contexts! And most certainly our personal design heuristics and adaptations of patterns for our specific design contexts mostly likely are not universal.

There is a difference between a single writer reporting 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 [KP]. For example, if a pattern has been observed multiple times in Java programs, does this necessarily imply that it will work for C++ C++ code as well? Without having observed or tested it, one cannot really (empirically) tell.

5.2 Stories about elephants

Stories deliberately remove information but try to bring order into a chaotic word. There are many ways to tell the story about elephants. And there are many ways to describe a pattern that we have experienced in the world. Our common description format, usually a variation of context-problem-forces-solution-consequences, is one way to describe elephants. It is a good way, indeed. It asks all the right questions.

Many people invoke the metaphor of a story or a play [Ris] and point out that patterns are not just about facts but should tell a story [Appl]. The context sets up the stage. As in a play, the forces are creating a tension and the solution is resolving the conflict—a happy ending.

However, is this sequence always the appropriate order? There are many ways to tell a story, and some stories start with end. If we were about to describe an elephant, would we naturally start with all his evolutionary history and reason about why this species fits into the very environment elephants live in? Or, would we start with a picture first and then go into details? Most accounts about elephants use the later approach. First show the object, and then explain the phenomenon.

If we consider the literature genre of patterns as storytelling, then we should allow and encourage different forms of telling this story: A short story; a whole book; a series of stories; or even telling the story with motion pictures or cartoons.

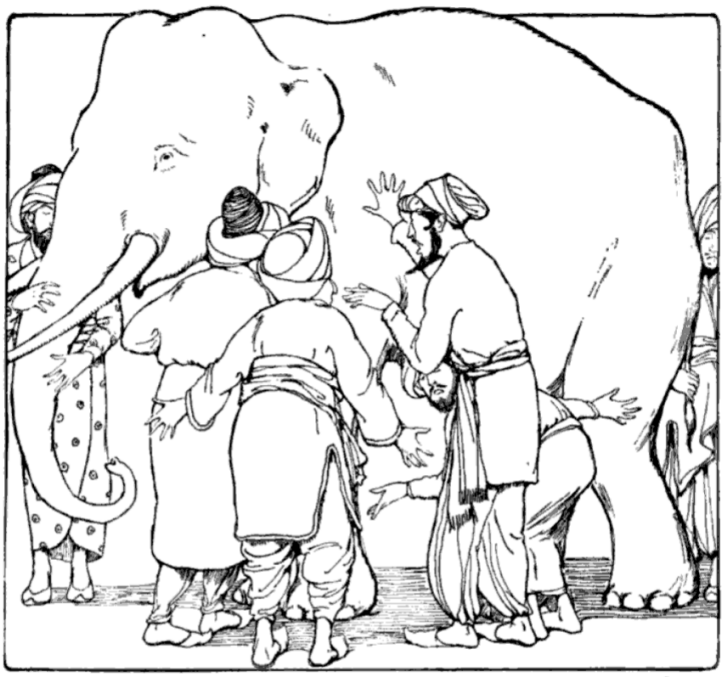
Stories are such a powerful tool because they are capable of transporting the wholeness of a solution. We experience wholeness if we follow a story in a novel. The plot unfolds chapter-by-chapter, paragraph-by-paragraph, sentence-by-sentence, and word-by-word. The parts make the story and the story gives meaning to each of the parts. A simple sentence such as “The door was locked” has its own

meaning; however, in the context of a larger story its meaning can shift. A locked door has a deeper meaning in a crime story where a victim tries to escape. The same sentence can have a different meaning in a love story: “She wanted to tell him her feelings and caught up with the train at the local station. The door was locked.” The context not only changes the meaning of the sentence; the single sentence that reveals an important fact or event can also change the meaning of the whole story. The story directs the development of the events, scenes and characters; at the same time the story is made up exactly out of these interrelated parts.

5.3 The story of the blind men and the elephant

The story of the blind men and the elephant is a very old parable that discusses the limits of perception and the meaning of context. It can be found in Buddhist, Hindu, and Jain texts (see Wikipedia). The parable goes like this (from Wikipedia):

> A group of blind men heard that a strange animal, called an elephant, had been brought to the town, but none of them were aware of its shape and form. Out of curiosity, they said: "We must inspect and know it by touch, of which we are capable". So, they sought it out, and when they found it they groped about it. In the case of the first person, whose hand landed on the trunk, said "This being is like a thick snake". For another one whose hand reached its ear, it seemed like a kind of fan. As for another person, whose hand was upon its leg, said, the elephant is a pillar like a tree-trunk. The blind man who placed his hand upon its side said the elephant, "is a wall". Another who felt its tail, described it as a rope. The last felt its tusk, stating the elephant is that which is hard, smooth and like a spear.



By Illustrator unknown - From Martha Adelaide Holton & Charles Madison Curry, Holton-Curry readers, Rand McNally & Co. (Chicago), p. 108., Public Domain, https://commons.wikimedia.org/w/index.php?curid=4581243

What does this parable mean for pattern writers? We need to be aware that we may experience parts of a pattern but are still missing important aspects. We need to be careful to not call ourselves experts too soon. It also shows that we can report and describe the very same things in quite different ways when we focus on different parts of the whole.

5.4 Learning more about elephants

Elephants have been the same for a long time. Even though their environment has changed, elephants have not changed, as evolution is slow. It is true that elephants behave differently in the context of civilization. However, this has been in their nature for much longer time. If there had been zoos and circuses 10,000 years ago, then elephants would have reacted in the same sad way as they do today. It is not elephants that have changed; it is their environment.

However, we continue to learn new things about elephants. We get a better understanding of elephants each day we continue to research them. New scientific tools (such as tracking of traces, scales, x-rays etc.) can provide new answers and information. However, all this information was already in the formation of elephants since the species emerged. It is only now that we unlock this information.

In the same way we can learn more and more about a pattern even if the pattern does not change. Hence, we should continue to do research on existing and established patterns. Do they work in all contexts? What other consequences are there? What other heuristics are there for solving a similar problem that haven't been written down as patterns?

Unlike elephants, however, patterns in the software industry are less stable. New technologies can make some patterns obsolete. When memory and CPU time was scarce, patterns were needed to optimize data size and performance. Today, in most contexts, it is more important to optimize for flexibility and robustness. Hence, some patterns need to be replaced or updated.

Takeaways

The patterns are out there, yes. But we need to understand that we only know parts of the whole story. Our ways of observing and analyzing good designs are limited. Never assume that you know everything about a pattern. There is always more to it.

The best we can do is to tell a story about the patterns. Such a story unfolds the inner relations of the wholeness of a good design. However, stories are never complete. Each stories has holes. Sometimes we are cutting out facts for convenience or to make the parts fit.

A story can be re-told in many different ways, with many variations, different levels of details, and commentary. A story can also develop. Parts can change. New facts emerge. Elements that become obvious over time can be left out.

6. Some conclusions and implications

Elephants may evolve slowly, but they react differently in different contexts. So, too, do patterns and other design heuristics. In addition to finding better ways to organize patterns and presenting relevant depictions of their use, we need to find better ways to explain how to adjust patterns into specific contexts, how to sort through them, and add or find the information we need when we are able and willing to absorb it. No pattern depiction is ever complete, nor should it be our goal to make complete descriptions.

We have argued that pattern descriptions try to capture forms that exist in the world. Whether a pattern description adequately captures a structure that can actually be found in the world is an outstanding question. Moreover, if the authors are not experienced in their domain they might capture the wrong patterns. But even if we have adequate patterns in our head, the explicate description will always be incomplete, misleading, or even contain the wrong elements.

Therefore, we need ways to ensure that the pattern descriptions actually represent meaningful patterns of the world. A written pattern should be the result of thoughtful pattern mining, a process that extracts "nuggets of wisdom." We can say "wisdom" because the insights in those patterns are grounded in many reviews of actual designs. Pulling out this knowledge is like mining for nuggets (Rising, 1998); the core of the pattern is pointed out; the noise of actual instances is taken away. This mining process is a process of cognition—as is any theory building. A pattern author often reconsiders the artefacts and examples her pattern is based on. Writing down the pattern, is also an active process whereby the writer tries to assemble the universal structure of the pattern. Pattern descriptions are proposals of specific

views on the world, and on solutions to agreed-upon design problems in particular. However, their validity needs to be tested, as does any theory.

Arnheim argues that the kind of concept created by contextually viewing objects is better suited for reasoning about those objects in different situations and under different conditions. Our current written forms for software patterns fall short in this regard—pattern depictions typically describe just enough context and forces, before providing a stylized, exemplary solution. For the most part, pattern solutions present a contextually isolated view. While this facilitates definition, classification, and learning, it does not build in the mind of the reader a deep understanding of that pattern in a realistic setting. And, even more challenging, in order to skilfully apply a pattern, a designer needs to be able to adapt a pattern to her specific context.

Rather than drowning pattern readers in even more text, verbal descriptions, and caveats, we propose that a better way to establish richer, more productive views of patterns would be to present curated views depicting multiple instances of particularly useful patterns in situ. Additionally, designers should be encouraged to create and potentially share notes on the heuristics they've used to adapt patterns to their specific contexts. How to best accomplish this (and which patterns warrant such curation) is a topic for future research.

To get the "right" views on patterns is problematic. Perhaps this shouldn't be our goal. Instead, we might seek augmented ways to depict software patterns that allow for productive thinking and their creative application. There are many ways of seeing the world and organizing its structure; and there is always doubt as to whether we have seen enough of the world to identify sufficiently stable patterns (let alone good examples of them). Patterns that have been identified in a pattern mining process are fallible in principle and can be falsified empirically. If a pattern consistently fails, it needs to be rejected—perhaps the pattern description, or the pattern in its entirety. However, patterns, like all design heuristics, are fallible. Successful pattern applications, on the other hand, are corroboration of the adequateness of the insight provided in a pattern description.

However, pattern descriptions, or any other account about design heuristics, are not simply about finding the "truth" about good design. They are also design tools to generate new good application. Thus, they go beyond ordinary theories. The quality of writing on patterns and the ways patterns are depicted matters as much as does the adequateness of the identified pattern.

References

[Alex79] Alexander, C. (1979). The Timeless Way of Building. New York: Oxford University Press.

[AISJFA77] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). A pattern language. New York, USA: Oxford University Press.

[Appl] Appelton, B. (2000). Patterns and Software: Essential Concepts and Terminology. http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html. (1.7.2009)

[Arn] Arnheim, R. *Visual Thinking,*

[Baey] Von Baeyer, H. C. Information: The new language of science. 2004. London: Phoenix.

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

[Good] Goodman, N. *Language of art: An approach to a theory of symbols.* Indianapolis, Ind: Hackett Publishing Co. 1976

[Koen] Koen, B.V. *Discussion of the method: Conducting the Engineer's approach to problem solving,* Oxford University Press, 2003.

[Koh12] Kohls, C. *The Path to Patterns - Introducing the path metaphor.* EuroPLoP 2012. – 17th European Conference on Pattern Languages of Programs. 2012. New York: ACM.

[Koh14] *The theories of design patterns and their practical implications exemplified for e-learning patterns.* 2014 https://opus4.kobv.de/opus4-ku-eichstaett/files/158/kohls_patterns13032014.pdf

[KP] Kohls, C., & Panke, S. Is that true? Thoughts on the epistemology of patterns. Proceedings of the 16th Conference on Pattern Languages of Programs. 2009. New York: ACM.

[Evan] Evans, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software,* Addison-Wesley, *2003.*

[Ris] Rising, L. (1998). The Pattern Handbook. Cambridge: Cambridge University Press

[Wirf17] Wirfs-Brock, R., "Are Software Patterns Simply a Handy Way to Package Design Heuristics?", PLoP 2017, Proceedings of the 23rd Conference on Pattern Languages of Programs.

[Wirf18] Wirfs-Brock, R. "Traces, Tracks, and Trails: An Exploration of How We Approach Software Design", PLoP 2018, Proceedings of the 24th Conference on Pattern languages of Programs.

[Whit] Whitehead, A. *Process and Reality.*