Advanced Pattern Writing

Patterns for Experienced Pattern Authors

Neil B. Harrison
Avaya, inc.
nbharrison@avaya.com

Introduction

Ten years ago, a group of us met at Allerton Park in Illinois and began a tradition of unique conferences, dedicated to creating a body of software pattern literature. That's a tall order, but we have done well. We have created a significant body of well-written patterns. They may not be literature yet, but they have improved much over the years. In short, we write a lot better than we used to.

It's time to raise the bar again. Some have criticized patterns for being too fluffy – lacking in content and substance. I fear that in some cases they may be right. As a result, I wrote these patterns. I offer them in the hope that they will help authors improve their patterns, particularly with respect to the substance in them.

These patterns are for two closely related audiences, the authors and the shepherds. Why have one set of patterns for two audiences? The obvious reason is that authors and shepherds are often the same people, and much of the advice is the same. But more importantly, an author can gain great insights by understanding what a reader – in this case, the shepherd – is looking for. So I modified the pattern format slightly. Each pattern has two solution summary statements. The main one is for the author, and the second one is especially for the shepherd.

These patterns form a little language. There are several possible paths through the language, but here is probably the most common sequence of the patterns:

"WHAT"-SOLUTIONS: *Write the core idea of the pattern in one or two key sentences that capture what the pattern is all about.*

"HOW"-PROCESS: *Extend the core idea with a detailed explanation of how to implement the solution.*

"WHY"-PROBLEMS: *Make the problem statement explain the main consequence of not having the solution in place.*

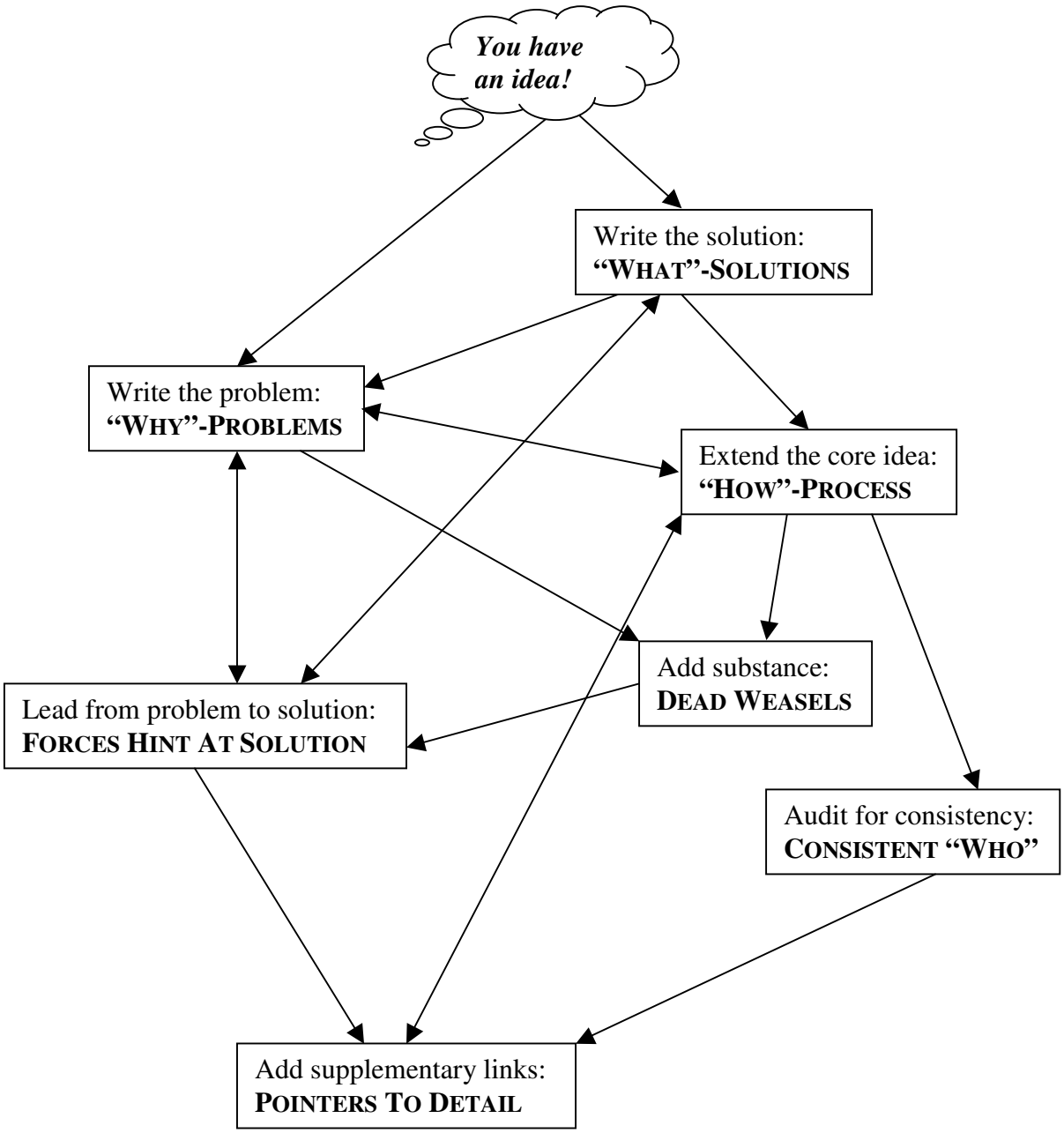
DEAD WEASELS: *Replace words that imply meaning but have little substance with a full, specific explanation.*

FORCES HINT AT SOLUTION: *Write forces so that they lead the reader from the problem to the general direction of the solution.*

CONSISTENT "WHO": *Make sure the pattern language focuses on the same audience throughout.*

POINTERS TO DETAIL: *Point to other sources for details that aren't germane to the solution, especially previously published material.*

Here are a few more paths through the language. As you can see, there are many ways to use these patterns together!



Note: lines indicate typical logical flows from one pattern to another. Cycles show the iterative nature of pattern writing.

Existing Work

These are not the first patterns to have been written about pattern writing and shepherding. There are two bodies of patterns that this pattern language builds on. The first is "A Pattern Language for Pattern Writing", by Gerard Mezaros and Jim Doble [Mezaros]. The pattern thumbnails relevant to these patterns are given below. Note especially that there are three patterns about naming patterns; there is no additional pattern in the present language about pattern names since these three cover the topic thoroughly.

Evocative Pattern Name: Choose a pattern name that conjures up images that convey the essence of the pattern solution.

NOUN PHRASE NAME: Name the pattern after the result it creates.

MEANINGFUL METAPHOR NAME: Find a meaningful metaphor and name the pattern after it.

SKIPPABLE SECTIONS: Clearly identify the Problem, Context, and Solution parts so that the readers can quickly determine whether this pattern applies to them.

VISIBLE FORCES: Regardless of the style chosen for the pattern description, ensure that the forces are highly visible.

CLEAR TARGET AUDIENCE: Identify a clear target audience and keep this audience in mind while writing the pattern.

TERMINOLOGY TAILORED TO AUDIENCE: Use only those terms with which the typical member of the audience can reasonably be expected to be comfortable.

RELATIONSHIP TO OTHER PATTERNS: Read other pattern languages and describe the relationships to other patterns.

CODE SAMPLES: Provide one or more implementation code examples, written in a prevalent programming language, to illustrate the pattern concepts.

CODE SAMPLES AS BONUS: Ensure that the pattern can stand on its own; able to communicate its essential concepts even if the code examples were deleted.

READABLE REFERENCES TO PATTERNS: When referring to patterns within the body of your pattern, weave the pattern names into the narrative.

The following patterns are in "The Language of Shepherding: A Pattern Language for Shepherds and Sheep", by Neil Harrison [Harrison].

BIG PICTURE: Start by reading the problem and solution of the pattern to get the main idea, and to give feedback on the problem and the solution first. By themselves, the problem and the solution should give the big picture of the pattern.

THREE ITERATIONS: Plan the shepherding to take three iterations of comments from the shepherd to the author.

MATCHING PROBLEM AND SOLUTION: Read the problem and solution together to make sure they match. The solution must address the whole problem, but not more than the problem.

FORCES DEFINE PROBLEM: The problem statement should describe the visible manifestation of something wrong. The forces give substance to the problem, and insight into what is behind the symptoms.

In the text, these patterns are annotated with [Mezaros] or [Harrison] to clearly distinguish them from the patterns in this language.

1. “What”-Solutions

…well begun is half done.

❖ ❖ ❖

You know of a really neat solution. It looks like it would make a good pattern, but it isn’t a pattern (yet).

This is where most of us begin with a pattern. We have found a design, an implementation, or some other idea that has worked well for us in solving a problem. A pattern is an attractive way to convey that information to others.

Unfortunately, it’s often surprisingly hard to write a pattern, particularly to start writing it. You may have a really good idea, but writing it down is daunting.

One of the problems is that we geeks aren’t necessarily good writers. It’s hard to describe an idea that works well in software in a natural language.

Another problem is that a pattern needs not only a solution, but also a problem, and lots of other information besides. The breadth of knowledge required to write a complete pattern may intimidate us. But we have to start somewhere.

Therefore:

Write the core idea of the pattern as the first sentence or two of the solution. Try to capture the essence of the solution in these few words.

It is crucial that the essence of the solution be absolutely clear, unambiguous, and understandable.

The solution summary should go right at the beginning of the solution, or somewhere else where it is obvious, and can be easily picked out on first reading of the pattern. In the Alexandrian form, the solution summary is set in bold just after the “Therefore:”. In the GOF form[Gamma], the solution summary finds itself in the “Intent” section at the very beginning of the pattern.

The solution summary may be a statement that describes the structure of the solution. The solution summary in this pattern is such an example. Or the solution summary may be an imperative; an instruction to do something. Other patterns in this language show this style. Pick whichever one works for you. In general, the structural statement is harder to come up with, but produces a stronger solution.

Occasionally, you may be stuck on trying to come up with the core idea of the pattern. If so, see the hints to shepherds, below.

Shepherds: Find and read just the solution summary. It should give you the general idea of the pattern.

This is usually the place to start with shepherding. If the author doesn't have this right, then the rest of the pattern doesn't matter.

It is very often helpful to compare the solution summary to the name of the pattern. Names generally evoke the solution. If the name and the solution summary don't seem to match, ask the author about it. Usually, one or the other needs to change.

Ask the author to tell you what the pattern is all about in one or two sentences. Alternatively, ask the author what is the most important point of the pattern.

Occasionally an author will struggle with this. If so, you might try asking the author to explain how the pattern came about. That often uncovers the real point of the pattern.

❖ ❖ ❖

…Note how this follows Alexander's philosophy of a pattern being a "thing, and the process to build that thing." The first few sentences state what the "thing" is. Note that additional explanation may be needed to describe the "thing".

Note that this is half of the BIG PICTURE [Harrison] pattern. You will later write the problem so that the author can get the big picture of the pattern quickly. Note that you want to make the solution (as well as the problem) prominent, as described in SKIPPABLE SECTIONS [Mezaros].

As of yet, you haven't described "the process to build that thing." This is a critical element of a pattern; the reader needs information on how to implement the solution. This is where the pattern HOW-PROCESS comes in.

2. "How"-Process

…with the solution summary in hand, you can see the pattern taking shape. But this doesn't mean that the reader can use or even understand the pattern yet. The pattern, particularly the solution, needs additional substance.

❖ ❖ ❖

The pattern has the core of a solution, and perhaps a brief problem statement, but a person couldn't pick up the pattern and use it yet.

As the author of a pattern, you are already living in the solution space. Things that seem obvious or even trivial to us may be neither trivial nor obvious to the reader. Because the material is new to the reader, the reader often must be led step by step through the solution.

At the same time, though, you must help the new reader get a picture of what the pattern is all about. You can't allow details to obscure the main message of the pattern.

Therefore:

Extend the core of the solution with an explanation of the solution in more detail, giving instruction on what to do, how to do it, and why to do it that way. Put this information after the solution summary, set off from it, but with leading information to go from the summary to the explanation.

The second part of the solution describes the "process to build that thing." This is the information the reader needs to implement the solution.

Make sure your solution details include these elements:

- Explain how the solution balances the forces, and tell which forces are not addressed. This may include discussion of tradeoffs; for example size vs. performance. Note that depending on the order that you write the pattern, you may not have written any forces yet! So you may have to defer this part of the solution until after you write forces. In infrequent cases, writing the "how" of the solution actually calls forces to mind, so you may find yourself iterating between forces and solution. See the pattern "Forces Hint at Solution."
- Explain why this is a good solution; why it works. It is as important to understand why a solution works as to understand how it works. That is what distinguishes patterns from ordinary instruction sheets. Some patterns put this in a section called "Rationale", but I prefer to weave it together with explanation of the resolution of the forces. The two are inextricably intertwined.
- Provide implementation hints for the solution. It is often a very good idea to draw a picture of it. Many patterns have sections called implementation. Structure diagrams or message sequence charts may be appropriate. This often includes recommended steps for implementing the solutions. This is usually the largest part of the solution.
- If the implementation has different options, provide hints for choosing which option. For example, the paragraphs following this list describe options about the order of writing the solution, and hints about which option to choose.

- Provide references to other patterns that may be needed.
- Provide pointers to additional details that do not contribute directly to the understanding of the pattern or its core issues (see POINTERS TO DETAIL).
- Discuss alternative to, and alternative implementations within the solution, as appropriate.

Which do you write first, the summary or the process to create it? While I tend to recommend writing the summary first, and then the process, the real key is your own preferences. Many people in software patterns think intuitively. They think of the big picture first. Process details come later, and come in the context of how they relate to the big picture. In this case, write the solution summary first, and then the process. Be careful not to skimp on the process details!

Other people's thinking is more oriented toward details. They derive the big picture (when forced to) from the details. If you think this way, it is probably much easier to write the implementation process, and then the solution summary. Be careful that your solution summary captures the essence of the solution, and isn't just a process step!

Shepherds: After reading the solution summary, read the entire solution to learn how to achieve the idea given in the summary.

The easiest place to start is to ask the author how to implement the solution. Or you can simply request more detail.

Don't forget to check to see that the summary matches the solution details. This usually isn't a problem, but demands immediate attention if it is.

In some cases, the solution may lack credibility. Refer to DEAD WEASELS for help here.

❖ ❖ ❖

…The result of a carefully crafted solution is that it contains the following parts:
- A sentence that gives the fundamental idea of the solution. This is not the structure to be created, but its intention.
- The concrete thing to be built. In Alexander's words, the spatial configuration of the elements and their cooperation.
- The process of how to create the structure.
- An explanation of why this solution resolves the core problem and balances the forces.
- A discussion of what the solution does not cover, such as unresolved forces.

The other result may be that the pattern becomes bloated.

3. "Why"-Problems

…we write patterns because we have a neat solution. But because of that, we naturally have trouble remembering the problems behind the solutions. This is illustrated in the saying, "To someone with a hammer, everything looks like a nail."

❖ ❖ ❖

Many pattern descriptions are colored by the solutions they suggest, which result in patterns that are tautologies. This is not particularly helpful to the reader, who has a problem, not a solution.

In far too many immature patterns, the problem and solution are basically restatements of each other. In other words, the pattern really has no problem. It may have something that is labeled as a problem, but when you put it next to the solution, they pretty much match.

A real example, excerpted from the pattern MIND YOUR MANNERS ([Rising1], used by permission):

> **Problem**: When we interact with customers, we don't always think about etiquette, dress, and behavior.
>
> **Solution**: Mind Your Manners. Be polite. Be aware of body language. The way you dress and behave influences the way the customer sees you and our company.

Clearly, this pattern contains very valuable advice. But the problem statement does not give any insight to the real problem; it is actually a reflection of the solution. It says that we don't think about etiquette and appearance, and the solution says to think about them. The general form is as follows:

Problem: You aren't doing X.
Solution: Do X.

Another common form is as follows:

Problem: How do you do X?
Solution: Do X.

Note that some patterns have this form of problem and solution, and are perfectly legitimate, but many such pairs are simply tautologies. Read the pattern carefully to determine whether or not it is a tautology.

The major reason this happens is that once we have a solution that we are excited about, it colors our thinking. It becomes difficult to imagine struggling without the pattern – it is just obvious that everyone should be able to see that this pattern is really great!

So we have to consciously step back and think about what the problem really is; why the solution is important.

Therefore:

The problem statement should reflect reasoning about the main consequence of not having the solution in place.

A good way to start is to ask yourself how the world would be worse if you don't use this pattern. Of course, you make "the world" specific to your pattern: if it is an OO design pattern, ask how your design would look if you didn't use the solution.

In the example above, imagine that you go into a sales call and don't pay attention to appearance and mannerisms. What is the consequence? Perhaps it is that the customer gets annoyed or offended by your actions, appearance, or odor. This draws their attention away from the nifty product you are selling, and they don't buy from you. It shouldn't be, but it is. Here is a possible improved problem statement:

> Successful business, be it acquiring a new customer, finalizing a new contract, or getting your ideas accepted in an organization is not just a matter of the quality of the offering. Often, the success or failure of a business is decided within the first 30 seconds, simply based on whether or not the other party likes the way you look.

It is sometimes helpful to ask yourself "why" questions about the solution, specifically why the solution is significant, or why one would want to apply the solution. You may find that you have to ask why several times to get to the heart of the problem.

Shepherds: read just the problem and solution, and see how similar they are. See whether the problem points directly to the solution. If so, probe with questions to help the author unearth the real problem.

This is similar to MATCHING PROBLEM AND SOLUTION [Harrison]. That pattern points out that the problem and solution must be consistent with each other. This pattern explains what to do if the problem and solution are too similar to each other.

The questions are the same as those stated above for the author. The most effective is usually, "How would the world be different if you didn't have this pattern?"

Don't be surprised if the author doesn't understand right away. After all, the author is sitting firmly in the solution space, and everything has the rosy glow of the solution. You may have to ask more than once, and may have to use try different approaches, such as the "why" questions. This is yet another reason to plan for THREE ITERATIONS [Harrison].

❖ ❖ ❖

…the process of discovering the problem is akin to peeling layers of an onion. It may take several tries to arrive at the real problem, so keep peeling! The result of all this peeling is sometimes astounding: the problem doesn't seem to resemble the solution at all! Yet you can see how they relate, and that's very cool. But now you may need to write material that ties the problem and solution back together.

4. Dead Weasels

…the initial problem and solution are in place. But they are often much easier said than done.

❖ ❖ ❖

The solution sounds plausible, but it is unclear how to make it work. In fact, you wonder if the author really understands what to do.

These can be the most frustrating of patterns. They sound right. You know you should follow them. But when you go to apply them, you feel that you have been cheated: you can't get from step A to step B. The author just happened to gloss over that with a few slick words. You have just encountered weasel words.

Weasels are traditionally sly, underhanded creatures. Likewise, weasel words are sly: they convey some meaning, but under the façade, there is nothing there.

The following statements were taken from real patterns; see if you can spot the weasel word in each.

- Hire a good consultant.
- Carefully design the system.
- Create a user-friendly interface
- Build strong teamwork

What is a "good consultant?" Under what conditions would you want to carelessly design a system? The phrase "user-friendly interface" is better than the previous two, but still doesn't give the reader much help. After all, would you ever recommend creating a user-hostile interface? And what does it mean to build strong teamwork? It has even less meaning than building strong teams.

Therefore:

Read through your pattern to find weasel words; words which imply meaning but have no substance. In nearly all cases, replace the weasel word with a phrase or paragraph that is more specific than the word it replaces.

Most weasel words are easy to spot. Look for adjectives or adverbs; they are the most likely candidates for weasel words. Also look for phrases that are platitudes, or always desirable. When you find such words, read the phrase, and ask what point you are trying to make. Then make the necessary corrections. For example:

> "Hire a consultant with experience in working with geographically distributed teams. The consultant should have expertise with OO design, although this is of secondary importance."

In some cases, you can drop the adjective or adverb, or the phrase altogether. In extreme cases, the pattern may be too superficial to save. Throw it out.

Note that in a few cases, weasel words may be all right. For example, pattern names that contain weasel words may be acceptable. But in every case, at least consider replacing the weasel word with more substantive wording.

Shepherds: ask the author what each weasel word or surrounding phrase means. Then have the author replace the word.

For example, you might ask the author, "What do you mean by 'good consultant'?" Or you could ask, "How do you tell a good consultant from a bad one?" Another possible question might be, "What is the difference between carefully designing a system and carelessly designing it?" Or, "Under what conditions would I not want to carefully design my system?"

❖ ❖ ❖

…this causes patterns to grow, often substantially. Sometimes a pattern will get too big, and will become too awkward to use easily. Then you have to balance detail in the pattern with detail outside the pattern. One way to do this is to apply POINTERS TO DETAIL.

5. Forces Hint at Solution

…in the most interesting patterns, the problem and solution are not close in cognition. When we think about the problem, the solution does not come to mind at all. This can make the solution seem particularly elegant, but it can also make it difficult to understand just how the solution solves the problem.

❖ ❖ ❖

Though the solution matches the problem, you don't see how the two fit together. Even after you read the entire pattern, the solution isn't intuitive.

One problem is that if the solution doesn't feel like it intuitively fits the problem, you won't remember it. You will always have to go back to the reference; it will never become part of you.

We have often said that the emergent patterns are the ones where the problem and solution are separate in time or space, or in different conceptual areas. In other words, the action one takes is not directly connected to the problem. But it causes something to happen (emerge) that solves the problem. But that makes it difficult to understand how a solution solves the problem.

In these cases, the author has had a flash of insight, and the insight is the real heart of the pattern. Now the trick is to convey that insight to the reader. A problem and solution alone is simply not enough; they need to be tied together.

So that means that you need to prepare the reader for the solution. The logical place to do that is in the space between the problem and the solution, where the forces live.

Therefore:

Write the forces so they lead the reader from the problem, and point in the general direction of the solution. It is not necessary to be able to derive the solution from the forces. Instead, write the forces to be the foundation of a bridge to the solution.

This sequence of writing seems to work best:

1. Write a sentence or two for the solution first. Try to capture the essence of the solution, but don't go into much detail yet.
2. Write the problem in one or two sentences. If you aren't sure of the problem, write something down as a starting point.
3. Once you have those in place, you can fill in the forces between them. The first paragraph after the problem generally gives more detail about the gnarly-ness of the problem.
4. The paragraph just before the solution sets the stage for the solution.
5. Work both ends toward the middle.

Of course, every pattern is different. If you are like me, you will flit from the forces to add more detail to the solution, and back again. And of course, you will modify the problem and the solution statements as you go along.

Many patterns have bullet lists of forces. Such lists may benefit from a reordering. Begin with the forces that are most "problem-like"; that explain the problem in more detail, or that show the contradictions inherent in the problem. They may lead naturally to other forces. In any case, order the forces so that the forces most indicative of the solution end up just above it.

Shepherds: ask the author to trace a logical sequence through the forces. Look for places where there are large non-intuitive leaps.

In some patterns, it may be difficult to find the forces, in contradiction to VISIBLE FORCES [Mezaros]. In that case, you will have to work with the author to make the forces more obvious. This does not necessarily mean that the author has to create a section called "Forces" in the pattern. There are other ways to make the forces clear.
Doesn't this pattern contradict FORCES DEFINE PROBLEM [Harrison], one of the beginning shepherding patterns? No, forces do both; they help one understand the problem, but they also give subtle hints in general direction of the solution. Sometimes the author struggles with the problem; it just won't come. In that case, writing the forces can help illuminate the problem. After you write the problem, you will probably need to order the forces, and fill in any missing ones.

❖ ❖ ❖

…There are two things that go on here. The most obvious is that the pattern flows better, particularly between the problem and the solution. The second is that you as the pattern writer begin to think in terms of logical flow from problem to solution, rather than a list of forces to be filled in.

6. Consistent "Who"

…when we write patterns, we often try to make them useful to as many people as possible. We try to address every issue that might come up, so that everybody can understand and apply the pattern. But this makes the pattern a jack-of-all-trades – but a master of none.

❖ ❖ ❖

The pattern is hard to use, because the pattern is unfocused.

This is often because the author doesn't really know whom the pattern is for. So the author tries to make the pattern useful to everyone.

Shepherding and writer's workshops can actually make this problem worse. Writer's workshops tend to cause patterns to become more general; to appeal to a wider circle of people. The author may receive advice that the pattern should address needs of a certain audience in addition to the audiences already in the pattern. But this can dilute the focus of the pattern.

The habits of readers complicate this problem somewhat. A reader new to the pattern wants to find out as soon as possible whether the pattern applies to him or her. A combination of the problem and the audience of the pattern determine this.

One can easily state the intended audience at the beginning of a pattern or pattern language. This is very useful, but has two drawbacks. The first is that it is easy to drift from the intended audience; the flow of writing can easily pull us from our moorings. The second drawback is that you can't depend on the readers to have well-behaved reading habits. Some may skip the introduction and jump in at the first pattern. Others may just read the problems and solutions. There are even some who start at the beginning and read the whole thing sequentially! For each of these groups of readers, you need to catch their interest early in their reading.

A pattern may solve the right problem for the right person, but it does them no good if they don't read it.

Therefore:

Ask yourself who will use this pattern, and make the pattern focus on one audience. Make this clear at different points throughout the pattern. Be consistent.

There are a few key places where it is good to establish who the audience is. The first is the title of the pattern collection. In the ideal cases, screens the audience effectively. For example, the title "Smalltalk Scaffolding Patterns" [Doble] keeps the C++ and Java riffraff away. Another is the introduction to the work. Note that the introduction to these patterns talks about the audience. Within a single pattern, one might focus the audience in the context section, in the problem statement, and/or in the solution statement. These are the bold sections, if using the Alexandrian form. The pattern name is too short to establish audience focus.

It often isn't necessary to explicitly state who the target audience is. Or in some cases, a single word suffices to establish the audience.

This is similar to CLEAR TARGET AUDIENCE [Mezaros] and TERMINOLOGY TAILORED TO AUDIENCE [Mezaros]. This pattern explains various places to establish the audience, and it stresses the necessity of maintaining consistency throughout the pattern.

Shepherds: as you read the pattern, ask yourself whom the pattern is intended for. Although it may be stated explicitly, watch for cues in the pattern that the audience focus has shifted.

The biggest difference in audience is in experience level, either general or specific to a technology or technique. Watch to see if the pattern begins to assume experience levels that are inconsistent. For example, the pattern may begin to use acronyms or jargon that the stated audience wouldn't know. On the other hand, watch for excessive explanations and definitions of terms the audience should know already.

❖ ❖ ❖

…So why does this pattern have two audiences? Isn't this pattern in direct violation of itself? It isn't, for two reasons. The first reason is that all the patterns consistently address both authors and shepherds. The second reason is that shepherds are a subset of authors; they must know how to write patterns. Advice to the shepherds is good for regular authors to see, and vice versa.

7. Pointers to Detail

… In the patterns world, we exercise an "aggressive disregard for originality." Sometimes we take it too far.

❖ ❖ ❖

The patterns community has been justly criticized for rehashing previously published material.

Many patterns contain restatements of older material. The material may be previously published, or sometimes unpublished common knowledge. There are often good reasons for writing such material as patterns: in some cases, the previous material was not written well enough to be easily approachable by most readers; in other cases, the material fits into a larger whole such as a pattern language. The language would be incomplete without the duplicate patterns. But in many cases, the material has already been published.

Yet even if there are good reasons for repeating existing material, there are three significant problems with doing so. The first is that it makes finding and referring to patterns more difficult; in effect, it pollutes the namespace. For example, the index of *The Pattern Almanac* [Rising2] devotes more than a page to layers, and includes the following: Layers, Layered Architecture, Layered and Sectioned Architecture, Four-Layer Architecture, Three-Tier Architecture, Two Tier Architecture, Hierarchy of Control Layers, and more.

The second problem is that duplication simply looks bad to the outside world. It makes the patterns community look unprofessional.

The third reason is that if you put enough material in the pattern to make the pattern really useful by itself, the pattern will be long and probably pretty hard to use. On the other hand, if the pattern is succinct enough to be easy reading, it runs the risk of being missing important substance.

Therefore:

Balance needed detail with conceptual flow in the pattern by using pointers to details that aren't germane to the key points of the pattern, and particularly to previously published patterns.

The challenge is always what to put in the pattern and what to have pointers to. Remember that you want to get the main idea across. So work with flow – the main pattern should establish a conceptual flow. You don't want to break the conceptual flow with momentary dives into unwanted detail, or into unwanted tangents. So you dive into enough detail to firmly establish the idea, and use pointers to more detailed information.

We are not good judges of our own work, because we already have intimate knowledge of the subject. So we tend not to notice when our patterns are too superficial, since we know what is behind them. Similarly, we aren't bothered by breaks in the flow, because we already know what the flow is. The obvious answer is another pair of eyes.

Make judicious use of examples, including code examples. Examples should be used to help clarify a point, but excessive detail can actually distract from the main point. As a rule, the longer and more detailed the example, the later in the pattern it should be.

An alternate approach with examples is to use formatting to set them off from the main text of the pattern, so that the reader can skip them if desired. You can put examples in a sidebar, for example. Code examples are set off naturally by the typical format of the code. In these cases, make sure that the pattern can stand on its own without the example. See CODE SAMPLES [Mezaros] and CODE SAMPLES AS BONUS [Mezaros] for further information.

Of course, not all patterns need the same level of examples. Technical patterns, such as OO design patterns, simply beg for code examples. On the other hand, patterns of process and behavior, such as these, don't lend themselves quite so much to examples. But on the other hand, such patterns lend themselves to simplistic platitudes for solutions. In either case, experience and comparison to other similar patterns will help.

Where do you point? In general, favor pointing to somewhere in the work itself, such as an appendix, over an outside source. That way, the reader doesn't have to get another document or buy another book.

Are hyperlinks a good way to implement pointers to detail? Hyperlinks can be good as long as the primary medium of the pattern is electronic. With hyperlinks, try to get all the hyperlinks at the same level.

Thumbnails; one or two sentence summaries of patterns can often be very useful. The pattern READABLE REFERENCES TO PATTERNS [Mezaros], suggests weaving the name into the text to increase readability.

How do you handle using a previously published pattern? If it is part of a pattern, then start with a thumbnail of the pattern, and add more detail as necessary. Be sure to clearly explain the RELATIONSHIP TO OTHER PATTERNS [Mezaros]. However, don't rewrite a previously published pattern and slap a new name on it. If you need to write something up, write it up as an example usage of the pattern in your context. Note, for example, CONSISTENT "WHO" in this collection. It is similar to an existing pattern. So it explains how it differs, what it adds beyond that pattern. This issue might be a point of discussion at the writer's workshop.

Shepherds: Do a "two-pass" read of the pattern. In the first pass, read the problem and solution to get the essence of the pattern. Then read the complete pattern, and note where the detail detracts from the basic flow as established by the problem and solution.

Use your personal reaction to the pattern to give you cues about the level of detail. Are there sections of the pattern that you feel like skipping, or that put you to sleep? Are there places where you don't have enough information to do what the pattern suggests? These things may indicate excessive or insufficient detail.

Another thing to watch for during the two-pass read is whether the pattern is a duplicate of an existing pattern. The first clue will be in the first pass; it may sound suspiciously like some other pattern. In the second pass, you should be able to pick up the essential differences from the existing pattern. If you don't see them, ask the author how the pattern is different from the existing pattern. (Note, of course, that this requires a familiarity with the pattern literature; check the pattern literature and any domain-specific pattern websites.)

❖ ❖ ❖

…One of the purposes of patterns is to raise the context of our communication, which allows the communication to flow better. We can discuss that our software design should be based on MVC without delving into the details of the responsibilities of the Model, the View, and the Controller. That same type of flow works within a pattern itself, with the caveat that a pattern is meant to teach, so it may need to explain details more often.

Acknowledgments

Over the years, I have had the privilege of reading hundreds of patterns. All the authors of these patterns have helped me, through their patterns; learn what makes a good pattern. Those who have allowed me to be their shepherd, as well as those who have shepherded my patterns, deserve special thanks. My shepherd for this work was Frank Buschmann, who gave me outstanding advice on this work. Many thanks! Thanks also to the participants of my EuroPLoP 2003 writers' workshop, who gave me many helpful suggestions.

These patterns were partly inspired by a set of patterns I presented at MensorePLoP in Okinawa, Japan in 2001. The comments I received from my shepherd Jim Coplien, Linda Rising, Tiffany Winn, and the writers' workshop there provided some insights for this work.

Thanks to Linda Rising for graciously allowing me to use part of the pattern MIND YOUR MANNERS as an example of what not to do!

References

[Doble] Doble, J. and K. Auer: "Smalltalk Scaffolding Patterns," in Harrison, N., B. Foote, and H. Rohnert, eds., *Pattern Languages of Program Design 4*, Addison-Wesley 2000, pp. 199-220.

[Gamma] Gamma, E. et al: *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, 1995.

[Harrison] Harrison, N. "The Language of Shepherding: A Pattern Language for Shepherds and Sheep", available at http://www.hillside.net/patterns/EuroPLoP2001/shepherding.doc.

[Mezaros] Mezaros, G. and J. Doble: "A Pattern Language for Pattern Writing," in Martin, R., D. Riehle, and F. Buschmann, eds., *Pattern Languages of Program Design 3*, Addison-Wesley 1998, pp. 529-574.

[Rising1] Rising, L. "Customer Interaction Patterns," in Harrison, N., B. Foote, and H. Rohnert, eds., *Pattern Languages of Program Design 4*, Addison-Wesley 2000, pp. 585-610.

[Rising2] Rising, L. *The Pattern Almanac 2000*, Addison-Wesley, 2000.