MetaPatterns: A Pattern Language for Pattern Writing

Gerard Meszaros
Object Systems Group
e-mail: gerard@osgcorp.com

Jim Doble
Allen Telecom
e-mail: jdoble@msn.com

Abstract
This pattern language captures "best practices" of pattern writing as observed at PLoP-95. It describes, in pattern form, the characteristics which the authors and other reviewers in their review groups found most appealing about the patterns and pattern languages they reviewed. The patterns in this language fall into two broad categories: patterns which apply to all patterns, and patterns which apply to patterns within a pattern language. This language attempts to be an exemplar of the practices it proposes.

1.0 Introduction

The following *Patterns* were "mined" from the experiences of the reviewers at PLoP-95. After reviewing a large number of *Patterns* and *Patterns Languages*, the reviewers began to develop a collection of *Pattern*-writing techniques and approaches that were observed to be particularly effective at addressing certain recurring problems. It was quickly recognized that these techniques and approaches could themselves be expressed in the form of a *Pattern Language*[1]. The contributors invite other *Pattern* writers and reviewers to try these *Patterns*, comment on their usefulness, and to submit their own favorites.

Pattern Language History

Much of the reason behind the creation of this *Pattern Language* was the realization that there was a "*Pattern*" behind what the reviewers liked about widely disparate *Patterns* and *Pattern Languages*. As the number of *Patterns* reviewed went up, these observations led to hypotheses that were tested in subsequent *Pattern* reviews. If the hypotheses were shown to be true more often than not, we started referring to them by evocative names. (In the beginning, the names were all we had to communicate them, so they had to be evocative to be effective.) Comparing notes based on our respective review groups'experiences, we quickly determined that the kinds of practices that were being applauded were very much the same despite the independence of our review groups. While we did not keep detailed statistics, we only present *Patterns* here that emerged from this broad consensus. Many of these *Patterns* were made available to practitioners conducting *Pattern* writing courses and the feedback to the authors indicates that they have been very helpful. We now make these *Patterns* available to a wider audience for usage, validation and improvement.

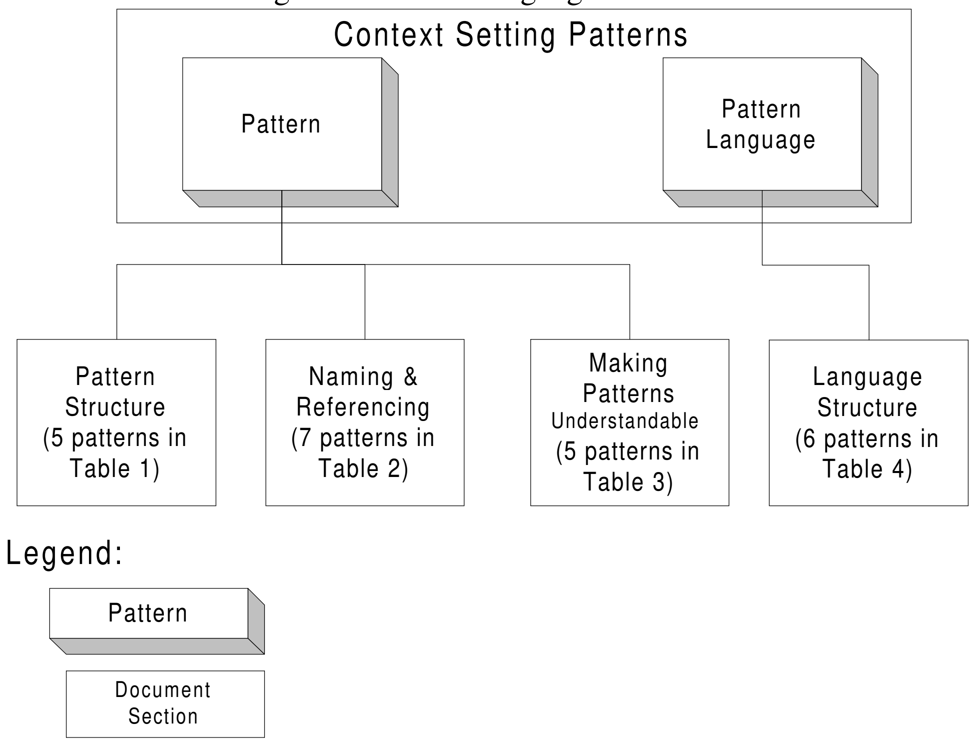
1.1 Pattern Language Summary

This *Pattern Language*, like Christopher Alexander's landmark work "A *Pattern Language*", is said to be "generative". Unlike a prescriptive *Pattern Language*, which describing the steps or recipes for solving some problem, this generative language describes the desired result. The

---

[1] Clearly the result of over-exposure to *Pattern Languages*. After two intensive days of reviewing *Pattern Languages*, just about any collection of scatter-brained ideas begins to look like a candidate for a *Pattern Language*.

Figure 1: Pattern Language Structure



Pattern author is free to employ different techniques to achieve these results. This approach should allow this *Pattern Language* to be employed in whole or in part as the reader sees fit.

The *Patterns* are grouped into five sections lettered A through E.

- Section A, **Context-Setting Patterns,** introduces the concept of a *Pattern* (a Solution to a Problem in a Context) and a *Pattern Language* (collections of *Patterns* that are related to each other by virtue of solving the same problems or parts of a solution to a larger, partitioned problem) so that they may be used throughout this *Pattern Language*.
- Section B, **Pattern Structuring Patterns,** contains *Pattern* describing the desired content and structure of individual *Patterns*, whether free-standing or part of a larger *Pattern Language*.
- Section C, **Pattern Naming and Referencing Patterns**, contains *Patterns* that describe techniques for naming your *Pattern*(s) and for including references to other *Patterns* within your *Pattern*(s).
- Section D, **Patterns for making Patterns Understandable** contain *Patterns* that capture techniques for making your *Patterns* and *Pattern Language*s easier to read, understand and apply.
- Section E, **Pattern Language Structuring Patterns,** contains *Patterns* that describe the desired the content and structure of *Pattern Language*s.

Each section starts with a brief summary which introduces the *Patterns* described in the section. The *Patterns* in the entire collection are depicted graphically in Figure 1 and summarized at the end of this paper in the *Problem/Solution Summary* section in tables 1-4.

1.2 How to Use These *Patterns*

The use of *Patterns* to communicate wisdom and insight in computer/software systems design is a relatively new idea. As such, techniques and approaches for writing *Patterns* and *Pattern Language*s are continually being improved, as creative individuals try new ways to organize and communicate their thoughts. There is no single right way to write a *Pattern Language*. These *Patterns* should be treated as suggestions to be tried and to be adopted where they help.

Throughout this *Pattern Language*, *Pattern* names have been highlighted typographically as have *Pattern* **terms**. Where possible, this language has been designed to exemplify the *Patterns* it expounds. In other words, the authors have attempted to practice what they preach by making it be its own *Running Example.* As such, this *Pattern Language* and the *Patterns* themselves can be used as examples, hopefully helping *Pattern Language* authors to organize and communicate their thoughts.

A reader searching for a solution to a particular problem should refer to the *Problem/Solution Summary* to see if any of the problems resemble the one they are trying to solve.

Each *Pattern* has been written with *Skippable Sections*; a reader trying to get a sense of the language should focus on the **Problem**, **Context** and **Solution** sections. Once they have determined that a *Pattern* is of interest, they can look at the **Forces** section for guidance on determining whether it is applicable to their situation.

A: Context-Setting *Patterns*

It is not the primary purpose of this *Pattern Language* to define the concept of a *Pattern* or a *Pattern Language*. But the *Patterns* in this language are applied within the context of writing *Patterns* or *Pattern Languages* thus we must include some sort of working definition. In keeping with the spirit of the *Patterns* movement, we do this in the *Pattern* form.

A.1 Pattern: *Pattern*

Problem:

How do you share a recurring solution to a problem with others so that it may be reused?

Context:

You are an experienced practitioner in your field. You have noticed that you keep using a certain solution to a commonly occurring problem.

Forces:

- Keeping the solution to yourself doesn't require any effort
- Sharing the solution verbally helps a few others but won't make a big impact in your field.
- Writing down your understanding of the solution is hard work and requires much reflection on how you solve the problem.
- Transforming your specific solution into a more widely applicable solution is difficult.
- People are unlikely to use a solution if you don't explain the reasons for using it.
- Writing down the solution may compromise your competitive advantage (either personal or corporate.)

Solution:

Write down the solution using the *Pattern* form. Capture both the problem and the solution, as well as the reasons why the solution is applicable. Apply *Mandatory Elements Present* to ensure that the necessary information is communicated clearly.

Rationale:

Knowledge is power. Sharing knowledge is sharing power. Making the information available to everyone by writing it down is the most efficient way of reaching large numbers of people both now and forever.

A.2 Pattern: *Pattern Language*

Context:

You are trying to describe a procedure with many steps or a complex solution to a complex problem. Some of the steps may only apply in particular circumstances. There may be alternate solutions to parts of the problem depending on the circumstances. A single *Pattern* is insufficient to deal with the complexity at hand.

Problem:

How do you describe the solution such that it is easy to digest and easy to use parts of the solution in different circumstances?

Forces:

- A single large solution may be too specific to the circumstance and impossible to reuse in other circumstances.

- A complex solution may be hard to describe in a single *Pattern*. A "divide and conquer" approach may be necessary to make the solution tractable.
- Factoring the solution into a set of reusable steps can be very difficult. Once factored, the resulting pieces may depend on one another to make any sense.
- Other *Pattern Language*s may want to refer to parts of the solution; they require some sort of "handle" for each of the parts to be referenced.

Solution:

Factor the overall problem and its complex solution or procedure into a number of related problems with their respective solutions. Capture each problem/solution pair as a *Pattern* within a larger *Pattern Language*. Each *Pattern* should solve a specific problem within the shared context of the language. Strive to ensure that each *Pattern* could conceivably be used alone or with a limited number of *Patterns* from the language.

To give the *Pattern Language* an identity of it's own, give it an *Evocative Name* by which it can be known and referenced. Describe the overall problem and how the *Patterns* work together to solve it in a *Pattern Language Summary*. Relate the *Patterns* to each other using *Readable References to Patterns* within the *Pattern* description, especially in the **Context** and **Related** *Patterns* elements.

Example:

This *Pattern Language* is itself an example of tackling the complex problem of writing *Patterns* and *Pattern Languages*. It presents the solution as a number of *Patterns* each of which describe the solution to a specific smaller problem.

B. Pattern Structure Patterns

A *Pattern* is just "a description of a solution to a problem found to occur in a specific context." But many other types of writing would claim to satisfy this phrase. What sets *Patterns* apart is their ability to explain the rationale for using the solution (the "why") in addition describing the solution (the "how"). A key contributor to this characteristic is the structure of the *Pattern* form.

The *Patterns* in this section describe the structure of an individual *Pattern,* whether free-standing or part of a larger *Pattern Language*. *Patterns* are easier to understand and apply when all *Mandatory Elements (are) Present* regardless of the *Pattern* style chosen. *Optional Elements When Helpful* gives *Pattern* writers considerable flexibility in what additional information they present and how they structure it to maximize the readers' understanding. While structuring your *Pattern*, strive to make it *Single-Pass Readable* to minimize the frustration of the *Pattern* reader. Some techniques for achieving this include having *Visible Forces* so they can be easily picked out, and *Skippable Sections* that can be bypassed to speed up the first reading.

B.1 Pattern: Mandatory Elements Present

Aliases: All Elements Present

Problem:

How do you make sure that all necessary information is covered in a *Pattern*?

Context:

You are writing a *Pattern*, either standalone or as part of a *Pattern Language*.

Forces:

- All *Patterns* do not require the same kinds of information to be effectively communicated. Capturing all elements regardless of need only clutters many *Patterns*.
- For a *Pattern* to be true useful, it must have a minimum set of essential information. These information elements are required to allow *Patterns* to be found when required and to be applied when applicable.
- If the necessary elements are missing, it becomes much harder to determine whether the *Pattern* solves the reader's problem in an acceptable way.
- There is no single correct style or template for *Patterns*; trying to impose one could stifle creativity and get in the way of effective communication.
- Readers expect certain information to be present in a *Pattern*. This is what differentiates a *Pattern* from a mere problem/solution description.

Solution:

Include the following Mandatory Elements in the *Pattern*. The exact names of these elements vary from one *Pattern* style to another and the exact order in which they appear in is not as crucial as ensuring that they are all present.

Pattern Name: A name by which this problem/solution pairing can be referenced.

Problem: The specific problem that needs to be solved. Use *Context-Free Problem* [2]to ensure that the problem is kept separate from the constraints on the solution.

---

[2]*Context-Free Problem* [yet to be written]: The problem statement should be independentof the Context. This allows other *Patterns* to be written that solve the same problem in different circumstances.

Solution: The proposed solution to the problem. Note that many problems may have more than one solution, and the "goodness" of a solution to a problem is affected by the context in which the problem occurs. Each solution takes certain **forces** into account. It optimizes some at the expense of others. It may totally ignore some **forces** if the context implies they are lower in priority. Use *Solution Clearly Related to Forces*[3] to ensure that the reader understands why this solution was chosen.

Context: The circumstances in which the problem is being solved imposes constraints on the solution. The **context** is often described via a "situation" rather than stated explicitly. Sometimes, the context is described in terms of the *Patterns* that have already been applied. The relative importance of the **forces** (those that need to be optimized at the expense of others) is determined by the **Context**.

Forces: The often contradictory considerations that must be taken into account when choosing a solution to a problem. The relative importance of the **forces** (those that need to be optimized at the expense of others) is implied by the **context**.

Rationale

A *Pattern* goes beyond a mere description of the solution by providing a window on the thought processes behind choosing the solution. The mandatory *Pattern* elements described here are essential to communication of this information. In the many *Patterns* that have been written since *A Timeless Way to Build* and *A Pattern Language* were first published, these mandatory elements have been found to be the minimum information required to effectively communicate a *Pattern*.

Examples

All the *Patterns* in this language have Mandatory Elements Present. This ensures that potential users of these *Patterns* understand why and when to apply them. The elements are highlighted through the use of headings. Some of these *Patterns* start with the **Problem** statement followed by the **Context**, while others (e.g. *Pattern Language*) start with the **Context**. This was done to illustrate both styles. [Bercz96] consistently places the **Context** before the **Problem** section.

In some *Pattern* styles, the *Pattern* elements have different names or are organized differently. See [Cope96] for a more complete list of *Pattern* styles and their features.

Christopher Alexander and his associates used this basic structure in *A Pattern Language*. The mandatory elements are separated typographically, with the **solution** paragraph(s) being introduced with a "therefore".

In *Design Patterns* [GHJV94], the Problem and Context sections are replaced with an Intent and an Applicability section, respectively which are augmented by a more concrete example of the problem in the Motivation section. The solution section is replaced by 4 sections: Structure, Participants, Collaborations and Implementation.

---

[3] *Solution Clearly Related to Forces* [yet to be written]: The solution section of a *Pattern* should clearly describe how the solution optimizes the forces that the context implies should be given the highest priority. This ensures that the reader understands how the forces influenced the selection of this solution.

B.2 Pattern: Optional Elements When Helpful

Problem:

How do you communicate essential information that does not fit well into the Mandatory Elements?

Context:

You are writing a *Pattern* and have applied *Mandatory Elements Present*.

Forces:

- All *Patterns* do not require the same kinds of information to be effectively communicated.
- Capturing all elements regardless of need only clutters many *Patterns*.

Solution:

The following sections may be included if they make the *Pattern* easier to understand or provide better linkage between the *Pattern* in question and related *Patterns*:

Resulting Context: The context that we find ourselves in after the *Pattern* has been applied. It can include one or more new problems to solve. This sets us up for applying more *Patterns*, possibly the next *Pattern*(s) in a language.

Related *Patterns*: Other *Patterns* that may be of interest to the reader. The kinds of *Patterns* include:

- Other solutions to the same problem,
- More general or (possibly domain) specific variations of the *Pattern*,
- *Patterns* that solve some of the problems in the resulting context (set up by this *Pattern*)

Examples: Concrete examples that illustrate the application of the *Pattern*.

Code Samples: Sample code showing how to implement the *Pattern*.

Rationale: An explanation of why this solution is most appropriate for the stated problem within this context.

Aliases: Other names by which this *Pattern* might be known.

Acknowledgments: You should acknowledge anyone who contributed significantly to the development of the *Pattern Language* or the techniques described in it. If your *Pattern* has been through a “shepherding process” or “writer’s workshop”, significant contributors (such as the shepherd!) are candidates for being acknowledged.

Examples:

In *Design Patterns* [GHJV94], the Resulting Context is known as Consequences and Code Samples is called Sample Code. Examples is called Known Uses and is augmented by a more concrete representation of the problem in the Motivation section.

B.3 Pattern: Visible Forces

Problem:

A *Pattern* presents a solution to a problem within a context. How do you ensure that the reader understands the choice of solution?

Context:

You are writing a *Pattern* or *Pattern Language* that is intended to convey one of potentially several solutions to a problem. You have applied *Mandatory Elements Present*; you are now writing the Forces section.

Forces:

- There are many different styles of *Patterns*, some more structured than others.
- People like to have convenient handles for concepts such as the forces which affect the choice of solution.
- Prose *Pattern* descriptions can be very pleasing to read but may be hard to use as a reference because the forces are buried in the prose.
- Having a separate Forces heading makes the forces very easy to find but may make the *Pattern* less pleasing to read.
- Too much structure can impinge on the literary quality of a *Pattern*.

Solution:

Regardless of the style chosen for the *Pattern* description, ensure that the forces are highly visible. This can be done by defining a meaningful "name" for each force and visually setting if off from text by making them minor headings, or by highlighting them using fonts, underlining, or other typographic techniques.

Example

This *Pattern Language* uses a **forces** heading to make the **forces** visible.

B.4 Pattern: Single-Pass Readable

Problem:

A person in search of a solution may need to look at many potential solutions. How do you make a *Pattern* quickly understandable to facilitate this search?

Context:

You are writing a *Pattern* with *Mandatory Elements Present*.

Forces:

- People sometimes only have a limited time to read a *Pattern*.
- People get frustrated and give up when the effort is too high.
- A *Pattern* that must be read several times before being understood is more likely to be misunderstood.
- A simple message is more likely to be understood correctly.

Solution:

Single-Pass Readable is easier said than done, and probably merits a *Pattern Language* on its own. There are a variety of techniques which can be helpful:

- Write the *Pattern* so that it is not necessary to read the later parts in order to understand the earlier parts. Use *Evocative Pattern Names* or *Pattern Thumbnails*[4] in cases where some understanding of a forward referenced *Pattern* is necessary for the reader to keep reading.
- Make it easier to pick out key information by using *Visible Forces* to highlight the tradeoffs involved. Use *Skippable Sections* (such as *Code Samples as Bonus*) to highlight information which can be skipped on first reading.
- In a *Pattern Language*, provide a clear, concise *Pattern Language Summary* outlining the structure of the *Pattern Language*, then remind the readers where they are within the structure as they go along, using *(Distinctive) Headings Convey Structure*.
- Observe the 7+/-2 rule[5]. Don't introduce more concepts in the beginning of your *Pattern Language* than the reader is likely to be able to remember, otherwise the reader will need to keep referring back to the beginning. Don't write the novel which introduces 50 characters in chapter one.
- Provide a *Glossary* so that readers don't have to search around for definitions of terms which they can't remember. This isn't *Single-Pass Readable* per se, but it is preferable to the alternative.

Rationale:

One of the greatest frustrations expressed by writers' workshop participants at PLoP-2 was with *Patterns* that had to be read several times before they could be understood. The *Patterns* that the reviewers found easiest to digest were all *Single-Pass Readable*.

Related Patterns:

- *Pattern Language Summary* introduces the larger problem being solved and how the solution has been factored into a number of *Patterns*.

---

[4] A brief footnote summarizing the main idea of a *Pattern*. Just like this one.

[5] Normal adults are able to keep seven (plus or minus two) items of information within their short-term memory, so say the Psychology textbooks.

- *(Distinctive) Headings Convey Structure* helps the reader understand the structure of the *Pattern Language* by reminding them where they are in the *Pattern Language*.
- *Skippable Sections* reduce the amount of reading required to understand the essence of the *Pattern* solution.
- *Evocative Pattern Names* and *Pattern Thumbnails* reduce the need to follow *Pattern* references before proceeding.
- *Glossary* defines the terminology in one place so the reader doesn't have to scan the *Pattern* (language) looking for it.

B.5 Pattern: Skippable Sections

Problem:

How do you minimize the amount of reading required to get the essence of a *Pattern*?

Context:

You are writing a *Pattern* that is part of a collection intended to be used as a reference. You have applied *Mandatory Elements Present* and are striving to make the *Pattern* easily understood and *Single-Pass Readable*.

Forces:

- The information required to determine whether a *Pattern* is applicable may be a small subset of the information required to actually apply the *Pattern*.
- People require different amounts of information to understand and apply a *Pattern*.
- People sometimes only have a limited time to invest in reading a *Pattern*.
- A long-winded *Pattern* description may cause a reader to skip the *Pattern* entirely because the expected return does not justify the investment.

Solution:

Clearly identify the **Problem**, **Context** and **Solution** parts so that the reader can quickly determine whether this *Pattern* applies to them. Put more detailed information (such as forces or code examples) in clearly identified sections that may be skipped if a person doesn't want all the detail.

Rationale:

Often, the hard part of finding a solution to a problem is discovering which *Patterns* solve your specific problem. This can require looking at a large number of *Patterns* to arrive at a small set of potential solutions. Much of the information in a *Pattern* is only required once you have narrowed down the list, or have decided to use the *Pattern*. This *Pattern* makes it easier to look at large number of *Patterns* in a shorter period of time.

Related *Patterns*

The *Pattern Code Examples as Bonus* describes a special case of a *Skippable Section***.**

Examples:

The "Alexandrian" *Pattern* style uses fonts and *** delimited paragraphs to allow the reader to pick out the problem and solution sections. More structured styles (such as used in this *Pattern Language*) use headings to separate the different sections. The introduction of this *Pattern Language* tells the reader which *Pattern* elements they should concentrate on for a "quick read".

In *Design Patterns*, the **Applicability** section allows the rest of the *Pattern* to be skipped when searching for a *Pattern* to solve a particular problem.

C. Pattern Naming and Referencing Patterns

Few *Patterns* live in isolation. Typically, they introduce new, hopefully smaller and more tractable problems which will lead you to other *Patterns*. Or, their may be other *Patterns* that solve the same problem. The *Patterns* in this section all deal with how to describe these *Relationships to Other Patterns* in a pleasing and efficient manner. Many elements of a *Pattern* may need to refer to other *Patterns*. Including *Readable References to Patterns* within the prose makes these references informative without being intrusive. Naming your *Patterns* with *Evocative Pattern Names* makes it easier to refer to them and reduces the need for the reader to follow the references every time you have included a *Pattern* name. Two especially useful *Patterns* for creating *Evocative Pattern Names* are *Noun Phrase Name* which names the *Pattern* after the resulting solution, and *Meaningful Metaphor Name*, which names the *Pattern* after some commonly understood metaphor. When the reader may need more information about the *Pattern* than just the name, include a *Pattern Thumbnail* as a footnote. A *Pattern*'s *Intent Catalog* is a good source for *Pattern Thumbnails*.

C.1 Pattern: Relationship to Other Patterns

Problem:

How do you make a *Pattern* part of a larger group of *Patterns*?

Context:

You are writing the **Related Patterns** element of a *Pattern* with *Mandatory Elements Present*.

Forces:

- Few *Patterns* are truly isolated; they usually lead to other *Patterns* or they solve problems set up by other *Patterns*.
- *Patterns* are more useful if their relationships to other *Patterns* are documented.
- Determining and describing the related *Patterns* can be hard work.
- Too many references to *Patterns* may distract the reader from the solution you are trying to describe.
- It may be hard to find all the related *Patterns*.

Solution:

While thinking about or writing a *Pattern*, read other *Pattern Language*s and identify relationships to other *Patterns*. Use *Readable References to Patterns* to cite related *Patterns*.

Rationale:

Understanding the relationships between a *Pattern* and other *Patterns* makes a *Pattern* more understandable and useful since alternative solutions can be assessed and follow-on *Patterns* can be applied.

C.2 Pattern: Readable References to Patterns

Problem:

How do you refer to other *Patterns* within the description of your *Pattern*?

Context:

You are writing a *Pattern* structured according to *Mandatory Elements Present*. You need to refer to other *Patterns* in one or more of the elements.

Forces:

- A *Pattern* may be described much more concisely if it can delegate much of its solution to other *Patterns*.
- Citations to other *Patterns* could disrupt the reader's train of thought or cause them to lose their context.
- The reader may want to read the *Pattern* being referred to and will require more information than just the name.

Solution:

When referring to *Patterns* within the body of your *Pattern*, weave the *Pattern* names into the narrative. Augment the *Pattern* name with a *Pattern* reference which can be used to look it up. Set off the *Pattern* name from the surrounding text by highlighting it typographically.

Where the *Patterns* have *Noun Phrase Names*, you should be able to use the *Pattern* name directly in the sentence (as we have done in this sentence.) In most cases, *Meaningful Metaphor Names* can be treated in the same way. *Verb Phrase Names*[6] are somewhat more difficult to weave into the narrative because they are typically "imperative" or "prescriptive" in nature; it may be harder to use them to describe the result.

To make it easier for the reader to find the description of the *Pattern*, it is desirable to include a reference to the *Pattern* description. This could take the form of a traditional literary reference of the form *Pattern-name*[reference-name], or you can use an *External Pattern Thumbnail*.

Rationale:

Weaving the *Pattern* names into the text makes the *Pattern* easier and more pleasurable to read while the reference satisfies those readers who want to find the original.

Related Patterns:

External Pattern Thumbnails can be used to refer to *Patterns* that must be understood to get the essence of the current *Pattern*.

Example:

In this *Pattern Language*, we have used a special character style for *Pattern* names to distinguish them from the surrounding text. Because most of the *Pattern* referenced have Noun Phrase Names, we have been able to weave the *Pattern* names into the text. The names have been augmented with a footnote or an internal *Pattern* reference number. We have included an *External Pattern Thumbnail* for any *Patterns* that are not included within the language.

---

[6] *Verb Phrase Name* [not included here]: name the *Pattern* after the process used to create the solution. Example: *Verb Phrase Name* is itself a Noun Phrase Name. To recast this as a *Verb Phrase Name*, you could try: *Name the Pattern After The Process Used to Create the Solution.*

C.2.1 Pattern: (External) Pattern Thumbnail

Context:

You are writing a *Pattern* that makes reference to related *Patterns* that appear later or are not included within the *Pattern* or *Pattern Language* you are writing. A basic understanding of these *Patterns* may be necessary for the reader to fully understand your *Pattern*.

Problem:

How do you refer to other *Patterns* in a concise but meaningful manner with minimum interuption of the reader's "flow", so that the understanding of your *Pattern* is maximized?

Forces:

- Referencing external *Patterns* is an effective way to build upon, or relate your work to existing *Patterns* and *Pattern Language*s.
- Including the complete description of related external *Patterns* within your *Pattern* will make it too large, and will distract the reader from what you are trying to communicate.
- A basic understanding of related external *Patterns* may be necessary for the reader to fully understand your *Pattern*.
- Some readers will be familiar with the referenced external *Patterns*, while others will not.
- The most concise way to refer to an existing *Pattern* is to provide sufficient information for the reader to obtain the complete description of the referenced *Pattern*, typically using an author/year tag and a References section.
- A reader may not have the time (or energy) to obtain and read the complete description of the referenced *Pattern*, prior to finishing reading and understanding your *Pattern*. Even if the reader does this they will be significantly distracted from your *Pattern*.

Solution:

Include the *Evocative Name* of the external *Pattern* within the text. The first time the external *Pattern* is referenced, provide both an author/year tag and a footnote with a brief (one or two sentences) "thumbnail" description of the essence of the external *Pattern*. The thumbnail should provide just enough information about the external *Pattern* to maximize understandability of your *Pattern*.

Rationale:

Readers who are familiar with the external referenced *Pattern* should not be distracted by the thumbnail footnote. Other readers will be able to continue reading and understanding your *Pattern* after they have read the thumbnail. Readers who want to understand the referenced *Pattern* in detail should be able to obtain its complete description using the information in the References section.

Related Patterns:

- *Evocative Pattern Name* helps reduce the need for *Pattern Thumbnails*.

Examples:

Several of the *Patterns* in this language include *External Pattern Thumbnails*. For example, *Evocative Pattern Name* contains a thumbnail reference to *Buffalo Mountain*[8].

Episodes [Ward96] includes a table of *External Pattern Thumbnails* as an Appendix to help the reader understand the essential aspects of yet unpublished *Patterns*.

C.3 *Pattern: Evocative[7] Pattern Name

Aliases: *Understandable Pattern Name, Solution Revealing Pattern Name*

Context:

You are writing a *Pattern* (or *Pattern Language*) that may need to be referred to by other *Patterns* or *Pattern Language*s.

Problem:

How do you name a *Pattern* so that it is easy to remember and refer to?

Forces:

- *Patterns* may vary based on differences in problem, context, forces, solutions, etc. or any combination of these. Each combination may require a distinct *Pattern* name.
- People should be able to use *Patterns* as a vocabulary, i.e., the identity of the *Pattern* becomes a "word" in a person's design vocabulary.
- A name short enough to use as a noun in a sentence may not convey enough meaning to be understandable out of context.
- The most memorable *Patterns* are those who names conjure up a clear image of the solution.
- Cute but obtuse *Pattern* names may be meaningful to the writer but few readers will remember what they mean later.

Solution:

Choose a *Pattern* name that are likely to conjure up images which convey the essence of the *Pattern* solution to the target audience. Imagine using the name in conversations or referring to it from other *Patterns*. Test the name by having people unfamiliar with the *Pattern* description guess at what the *Pattern* might be proposing based only on the *Pattern* name.

Rationale:

As *Patterns* are used to construct systems or to express how they are related to one another, the name is used without the accompanying description. A name chosen using this *Pattern* is more likely to be understood which makes it more likely to become part of the vocabulary of the readers..

Related *Patterns*

Intent Revealing Method Selector in [Beck96] describes the solution to the problem of naming methods in Smalltalk programming. By capturing the intent in the method name, the reader of the program should not have to refer to the method description every time they see the name.

Two *Patterns* for creating Evocative *Pattern* names are: *Noun Phrase Name* and *Meaningful Metaphor Name*. *Buffalo Mountain*[8] is an example of a cute but obtuse name for a *Pattern* which many people remember but few can recall what it describes.

---

[7] Evocative: adj. Tending or having the power to evoke. Evoke: n. To call to mind or memory. (American Heritage Dictionary)

[8] *Buffalo Mountain* in [Cope94] describes how to optimize communication within an organization. "The *Pattern*'s name comes from the similarity between the visual graph (when the communications between roles in a project are plotted in a scatter gram) and the characteristic shape of a mountain in Colorado, and from analogies that can be made about the foreces contributing to each."

This *Pattern* can also be applied to *Pattern Language*s since they, too, require names which are memorable and easy to refer to.

Example

All the *Patterns* in this language have *Evocative Pattern Names*.

Many of the *Patterns* in well known *Pattern* books, such as *Design Patterns*, use names designed to invoke images of the solution: *Bridge*, *Adapter*, *Proxy*, *Decorator* to name a few.

C.3.1 Pattern: Noun Phrase Name

Aliases: *Solution Phrase Name*

Context:

You are writing a *Pattern* (or *Pattern Language*) that may need to be referred to by other *Patterns* or *Pattern Languages*. You have created a *Pattern* that you are attempting to name by applying *Evocative Pattern Name*.

Problem:

How do you name a *Pattern* so that it is easy to remember and refer to?

Forces:

- Names that describe the problem are not unique since there may be several solutions to the problem. Supposed you were directed to "Apply the *Implementing State Machine Pattern*." If there were several *Patterns* which solved this problem, which solution does this name refer to?
- Names containing verbs or prepositions are difficult to use in conversation. Note the tension in this conversation: "What's that?" "Oh, it's an Object from a State."
- Names that describe the process of creating the solution are hard to use in a sentence describing the solution. It forces you to use phrase like: "In this design, we have an example of Create Objects for States."
- Describing the result of applying a *Pattern* helps the reader visualize the result but it does not help convey the problem being solved. "What's that? Oh, that's a State Object."
- Since an important purpose of *Patterns* is to foster communication by creating a shared vocabulary, *Pattern* names should be easy to say.

Solution

Name the *Pattern* after the result it creates. This allows the name to be used easily in conversation. In a *Pattern Language*, use a *Problem/Solution Summary* to help the reader find the right solution.

When referring to a *pattern* with a *Noun Phrase Name* where an understanding of the problem is important, include the problem in the referring phrase as in: "In this design, we used a *Proxy* to allow an object to be referred to remotely.."

Rationale

One of the most compelling aspects of a *Pattern* is the way it transforms a situation, resolving some forces and giving rise to others. This is what makes *Patterns* more than just "design rules" or a "style guide".

Focusing on the thing created by a *Pattern* for naming leads to noun phrases. Naming the *Pattern* above after the object it creates results in the name "State Object".

Example

The name of this *Pattern, Noun Phrase Name,* is itself an example of a *Noun Phrase Name*.

Related *Patterns*

This *Pattern* is a way to create an *Evocative Pattern Name*.

Another way to create an *Evocative Pattern Name* is to use *Meaningful Metaphor Name.* A *Meaningful Metaphor Name* may itself be a *Noun Phrase Name*.

C.3.2 Pattern: Meaningful Metaphor Name

Problem:

How do you give a *Pattern* a useful and memorable name?

Context:

You are writing a *Pattern* (or *Pattern Language*) that may need to be referred to by other *Patterns* or *Pattern Language*s. You have created a *Pattern* that you are attempting to name by applying *Evocative Pattern Name*.

Forces

- Metaphors are a good source of short *Noun Phrase Names*.
- People often find it easier to understand new concepts if they can be related to other concepts with which they are already familiar.
- If you try to explain new concepts in terms of unfamiliar concepts, the reader will be baffled. Rocket Science metaphors are typically understood only by Rocket Scientists. Star Trek metaphors ("Darmok and Jalad at Tenagra") are best understood by Trekkers. Hockey metaphors are understood best by Canadians.
- If the link between the metaphor and your *Pattern* is clear, readers will be able to transfer their knowledge of the metaphor into the context of your *Pattern*, helping to clarify and facilitate their understanding of your *Pattern*.
- If the link between the metaphor and your *Pattern* is unclear, the reader will be baffled. "I understand rocket science, but what does that have to do with your *Pattern*?"

Solution

Find a meaningful metaphor for the *Pattern*, and name the *Pattern* accordingly. Some people are better with metaphors than others, so if a good metaphor doesn't jump out at you, go back to Noun Phrase Name. Ideally a metaphor will be familiar and easily understood by the average reader. If you have to explain the metaphor, it is not familiar enough. Clearly identify how the problem and solution relate to the metaphor, so that the reader is able to link his/her understanding of the metaphor concepts with your *Pattern*.

Rationale

A metaphor effectively creates an association between your *Pattern* and a set of parallel concepts with which the reader is (hopefully) familiar. Naming your *Pattern* according to the metaphor you use to explain it helps the reader remember both your *Pattern* and the metaphor. Clearly linking metaphor concepts with *Pattern* concepts will help readers transfer their knowledge of the metaphor into the context of your *Pattern*, helping to clarify and facilitate their understanding of your *Pattern*

Example

The *Shopper Pattern* in [Doble96] is an example of a *Meaningful Metaphor Name*. This design *Pattern* describes how a Shopper object visits a number of objects to fill its Shopping Bag with items specified in a Shopping list. The name evokes an image of a person wandering from store to store trying to gather all the items on their shopping list.

Related Patterns

This *Pattern* is a specialization of *Evocative Pattern Name*.

Noun Phrase Name is an alternative way to create an *Evocative Pattern Name* though many *Meaningful Metaphor Name*s are also *Noun Phrase Names*.

Buffalo Mountain[9] is an example of a metaphor name that is not clearly explained as part of the pattern description, thus it has meaning only for the author and those with whom it has been shared verbally.

This *Pattern* is itself an example of *Duplicate Problem Description*[10] since it has the same problem statement as *Noun Phrase Name***.**

---

[9] *Buffalo Mountain* in [Cope94] describes … The *Pattern* is named after the shape created when … is plotted in a scatter gram.

[10] Described briefly in the Related Patterns section of E.2 Pattern: Common Problems Highlighted

C.4 Pattern: Intent Catalog

Problem

How do you make it easy to cross-reference *Patterns* in a meaningful way?

Context

You are writing a *Pattern* to which you would like other *Pattern* writers to refer.

Forces

- Coming up with concise summaries of problems and/or solutions is hard work; not everyone can do a good job of it.
- People will not provide references if it is too much effort.
- People who don’t understand your *Pattern* completely may make up inappropriate summaries of it.
- *Patterns* get too big if everything must be included.
- *Patterns* are hard to understand if relevant information is not included or referenced.

Solution

Provide a catalog of **pattern intents** that can be used as *Pattern Thumbnails* when other *Patterns* need to refer to this *Pattern*. The **intents** in the catalog should provide a 1-2 sentence "thumbnail" of what this *Pattern* does. Where the *Pattern* can be used to achieve more than one **intent**, each **intent** should be in the catalog.

Resulting Context

Patterns that are easier to refer to from other *Patterns*. This will result in more cross-referencing being done because it is easier to do.

Rationale

A good way to encourage something to happen is to make it the path of least resistance. Providing the intent catalog makes cross referencing less work than duplicating the information.

Related *Patterns*

The result of this *Pattern* may be used as an *External Pattern Thumbnail*.

Examples:

In this *Pattern Language*, the *Intent Catalogs* of all the *Patterns* have been collected into a *Problem/Solution Summary* table as part of the *Pattern Language Summary*.

D. Patterns For Making Patterns Understandable

A *Pattern* is only as useful as it is perceived by its users. *Pattern* writers can put a lot of effort into describing their *Patterns* but all this effort is for naught if the reader cannot understand it or gives up out of frustration. A key contributor is the quality of the writing, a factor into which we will not delve here. There are other factors some more specific to *Patterns*, others somewhat general. The *Patterns* described here are included because the reviewers felt they made a significant difference in how easily a *Pattern* or *Pattern Language* was understood.

The *Patterns* in this section apply to all elements of a *Pattern* or *Pattern Language*. They all strive to help the *Pattern* writer communicate their thoughts to the *Pattern* reader in the most effective manner possible. A key step is the identification of a *Clear Target Audience*. This helps the writer chose *Terminology Tailored to Audience* as well as commonly *Understood Notations* for diagrams and illustrations. If the audience includes programmers, it is appropriate to include *Code Samples*, while *Code Samples as Bonus* ensures that the reader isn't obligated to read them to understand the *Pattern*.

D.1 Pattern: Clear Target Audience

Aliases: *Target Audience, Identified Audience*

Problem:

Many people may read a particular *Pattern*. How do you ensure that a *Pattern* is easily understood by its intended audience?

Context:

You are writing a *Pattern* or *Pattern Language*.

Forces:

- A *Pattern* can be different things to different people.
- You can't satisfy all the people all of the time.
- Different people use different terminology.
- People with different backgrounds require different amounts of detail.

Solution:

Clearly identify a primary target audience with whom you would like to communicate the solution. Keep this audience in mind while writing the *Pattern*. "Test" the *Pattern* with (representative) members of the target audience.

It may even be useful to explicitly describe the target audience in the *Pattern* (language) introduction. This helps set the expectations of the reader by telling them "up front" that they are (not) the intended audience. It also helps people determine which meaning of an ambiguous term you had intended.

Rationale:

A *clear target audience* focuses the *Pattern* by providing criteria for including some information in the *Pattern* and omitting other information.

Related Patterns:

Once you have identified the target audience, choose *Terminology Tailored to Audience* to maximize the bandwidth of communication to them.

Example:

This *Pattern Language* was written specifically for people interested in writing *Patterns*. It does not explain basic concepts of what a *Pattern* is but it does explain at length the rationale behind why certain parts of a *Pattern* are important.

D.2 Pattern: Terminology Tailored to Audience

Problem:

How do you maximize the likelihood of the intended reader understanding your *Pattern*?

Context:

You are writing a *Pattern* or *Pattern Language* and have identified a *Clear Target Audience*.

Forces:

- Concepts can be described using a variety of language styles and terminology.
- Translating abstract concepts into concepts within a specific domain may be difficult for some people. The more concrete the terminology, the more likely it is to be understood by people familiar with the terminology.
- The goal of a *Pattern* is to be useful to the reader. If the reader doesn't understand the terminology, the *Pattern* will not be as useful.
- Expanding all acronyms and technical terms makes a *Pattern* description more wordy.
- Using terms without defining them can lead to misunderstandings (i.e. false agreement).

Solution:

Use terminology that is tailored to the audience. Use only those terms with which the typical member of the audience could reasonably be expected to be comfortable. Test the terminology with representative members of the target audience. As part of the introduction be sure to inform the reader of the "default" terminology source.

Rationale:

A *Pattern* that can be understood by the target audience is more likely to be useful.

Example:

This *Pattern Language* uses terminology specific to the *Patterns* community. It does not explain terms such as **Forces** and **Context** because the *Target Audience* is already familiar with them and would be put off by detailed explanations.

D.3 Pattern: Understood Notations

Context:

You are writing a *Pattern* and are trying to communicate concepts that are most appropriately communicated using diagrams or illustrations. You have identified a *Clear Target Audience*.

Problem:

How do you ensure that the diagrams are easily understood by your entire target audience?

Forces:

- Diagrams and illustrations are often more effective than prose when it comes to communicating concepts, especially those related to software design. "A picture is worth a thousand words."
- For any given concept (e.g. object model relationships) there may be a variety of diagramming notations and styles that can be used (e.g. Booch, OMT, etc.).
- Readers are not necessarily familiar with all such notations and styles. If the readers are not familiar with the notation you have used, they may be unable to understand your *Pattern*. "A picture you can't understand is worth a thousand words you can't understand."
- Readers are diverse. If you leave room for interpretation, different readers may interpret your diagram in different ways.
- Providing a detailed description of diagramming notations to your *Pattern* will make it too large, and will distract the reader from what you are trying to communicate.
- An expressive but obscure notation is less effective at communicating with most audiences than a less expressive but better-known notation.

Solution:

Use diagramming notations that are likely to be familiar to the target audience. Such notations should be widely used and easily understood (e.g. message sequence charts). If you are using a standard notation, always provide a reference to the standard. If not, or if there is any likelihood that potential readers are not familiar with the notation you are using, provide a clear, concise explanation of the notation when you first use it or refer the reader to a more detailed explanation in an appendix.

Rationale:

The more widely used the notation you use, the more likely that readers will be able to understand your diagrams without the need for a bulky and distracting explanation. Brief explanations of less common notations will help readers who are unfamiliar with the notation understand your diagrams, hopefully without creating too much of a distraction from the essence of your *Pattern*. References for standard notations provide a means for curious readers to learn more.

Related Patterns:

- If the explanation is included within the *Pattern* or *Pattern Language*, ensure that it is a *Skippable Section*.

D.4 Pattern: Code Samples

Aliases: *Code Examples*

Context:

You are describing a solution to a software architecture or design problem. You have identified a *Clear Target Audience* that includes significant numbers of software designers and programmers.

Problem:

How can you make a software *Pattern* sufficiently clear and unambiguous to facilitate straightforward implementation?

Forces:

- Software-related concepts are often complex and difficult to explain.
- Informal descriptive text is often unclear, and ambiguous.
- Programming languages are designed to convey software concepts in a formal, precise, and unambiguous manner.
- Many software workers are experienced and adept at reverse engineering concepts from software examples, and in fact prefer to learn ideas by looking at code.
- Many software *Patterns* can be implemented in many different ways.
- Many software workers are able to understand a software concept from a single "reference" code example and tailor their own implementation according to the needs of the system they are designing.
- Too much code interrupts the pattern’s flow and may make it unmanageably large.

Solution:

Provide one or more implementation code examples illustrating the *Pattern* concepts. Use a programming language likely to be understood by the *Target Audience*.

Provide one or more implementation code examples, written in a prevalent programming language, to illustrate the *Pattern* concepts. Choose an implementation approach that clearly demonstrates the essence of the *Pattern* in a straightforward manner while minimizing unnecessary or distracting detail. Ensure that the code samples are well-commented and that all assumptions and design decisions are stated. Differentiate between aspects of the example that are essential to the *Pattern* vs aspects that are arbitrary.

Rationale:

Well-commented example code is formal, precise, and unambiguous, and can be readily understood by many experienced software workers. Code examples provide concept reinforcement, providing a means for the reader to verify that they have understood the essential concepts of the *Pattern*.

Example:

All of the *Patterns* in *Design Patterns* [GHJV95] include *Code Samples*.

Related Patterns:

Code Samples as Bonus ensures that the *pattern* can be understood without the *Code Samples* and can help reduce the disruption of flow.

D.4.1 Pattern: Code Samples as Bonus

Aliases: *Code Examples as Bonus*

Problem:

How can you ensure that the essence of your software *Pattern* can be understood by your entire target audience, regardless of their familiarity with specific programming languages?

Context:

You are writing a software architecture or design *Pattern* and are including *Code Samples*.

Forces:

- Well-commented example code is formal, precise, and unambiguous, and can be readily understood by many experienced software workers.
- There is no universally understood programming language. *Code Samples* will be understood only by those readers who are familiar with the language you use.
- *Patterns* need to be concise. You can't afford to include *Code Samples* in every language.
- It can be difficult to extract the “big picture” from masses of implementation code.

Solution:

Ensure that the *Pattern* can stand on its own, able to communicate its essential concepts even if the code examples were deleted. *Code Samples* should be treated as an optional bonus, providing concept reinforcement and implementation guidance for those readers who are familiar with the language of the examples. Ensure that *Code Samples* embedded within the text can be easily skipped, or that they are in a separate *Skipable Section*.

Descriptions of essential algorithms and key object relationships and interactions should be provided using notations other than implementation code. Suitable notations include: pseudo-code, flowcharts, object modeling notations, event traces and object interaction diagrams. Whatever notations you choose, ensure they are *Understood Notations*; don’t invent your own unless absolutely necessary.

Rationale:

The purpose of a *Pattern* is to communicate to as wide an audience as possible. If the *Pattern* cannot be fully understood without reading the code examples, then readers who are not familiar with the example language will not be able to understand the *Pattern*.

Code Example:

```
Reader >> understandsSmalltalk
        “Answers True if the reader understands Smalltalk, otherwise signals an exception”
        self understands: #Smalltalk
                ifTrue: [^True]
                ifFalse: [self doesNotUnderstand].
```

E. Pattern Language Structuring Patterns

This section contains *Patterns* that solve problems unique to *Pattern Language*s. They deal primarily with how to assemble a number of related *Patterns* into a cohesive *Pattern Language* that is more than the sum of its parts. The language should be introduced using a *Pattern Language Summary* that introduces the overall problem and the *Patterns* that will be used to solve it. The *Problem/Solution Summary* is a key part of this introduction because it allows individual *Patterns* in the *Pattern Language* to be picked out when the document is used as a reference manual. Larger *Pattern Language*s often have a non-trivial structure that can be better communicated using *Distinctive Headings (that) Convey Structure*. They also often contain alternative, possibly mutually exclusive solutions to the same problem; these can be pointed out by ensuring that *Common Problems (are) Highlighted*.

A good way to tie together the *Patterns* in a *Pattern Language* is through the use of a *Running Example* that illustrates the application of the *Patterns* to an example of the larger problem. To improve understanding, any non-standard terminology should be expanded in a *Glossary*.

E.1 *Pattern: Pattern Language Summary

Problem:

How do you give the reader an overview of a set of *Patterns*?

Context:

You are writing a *Pattern Language* describing the solution for a complex problem.

Forces:

- A *Pattern Language* should be more than just the sum of its parts.
- The connections between *Patterns* (how they relate with one another) are not always obvious.
- Inter-*Pattern* relationships are sometimes difficult to understand solely from the perspective of the *Patterns* involved in the relationships.
- Describing the relationships between many *Patterns* in one place takes extra effort and increases the bulk of the language.

Solution:

Identify the set of *Patterns* as a *Pattern Language* and write a summary which introduces the larger problem and the *Patterns* which contribute to solving it. This summary explains why the *Patterns* belong together, the common threads found in more than one *Pattern*, and how the *Patterns* can be used together to do something useful. It can also be used to introduce the *Running Example.*

In larger *Pattern Languages*, it is useful to provide a *Problem/Solution Summary* to help the reader find the *Pattern*(s) which solve their specific problems.

Rationale:

A *Pattern Language Summary* provides the “big picture” while the related *Patterns* section of each *Pattern* provide the detailed linkages. The *Pattern Language Summary* may be the only place one can talk about the *Pattern Language* as a whole.

In the PLoP-2 review sessions in which the authors participated, the reviewers consistently preferred languages that introduced the *Patterns* in a *Pattern Language Summary* over those that launched right into describing the *Patterns*.

E.1.1 Pattern: Problem/Solution Summary

Context:

You are writing the *Pattern Language* **Summary** of a *Pattern Language* that includes *Patterns* which may be useful individually as well as within the flow of the language. Many of the *Patterns* have *Noun Phrase Names* based on the solution.

Problem:

How do you make it easy for a reader to pick out useful *Patterns* that solve their problem?

Forces:

- The problems and solutions in a *Pattern Language* may be spread across many pages of text. It could be very time consuming to read the whole language in search of a particular (yet to be identified) *Pattern*.
- A person using a *Pattern Language* may not need to use all the *Patterns* in the language (and certainly not all at once).
- Summarizing *Patterns* in the introduction takes extra effort and increases the size of the *Pattern Language*.

Solution:

Provide a table in the *Pattern Language Summary* that summarizes all of the *Patterns,* including a brief description of each *Pattern*'s problem and the corresponding solution. You can do this by collecting the *Intent Catalogs* for all the *Patterns* in the language into a single convenient table as part of the *Pattern Language Summary* or in an Appendix or References section.

Rationale:

This additional information helps the reader of the *Pattern Language* quickly zero in on the *Pattern*(s) that may solve their specific problem.

Examples:

This *Pattern Language* use a *Problem/Solution Summary* to give the reader an early indication of the structure of the language.

E.2 Pattern: Common Problems Highlighted

Context:

You are writing a *Pattern Language* that provides several *Patterns* that solve the same problem.

Problem:

How do you make readers aware that they should choose one of the alternative solutions?

Forces:

- A *Pattern* is normally considered to be a problem-solution pair. Most *Pattern* forms currently in use do not lend themselves to sharing a problem section amongst several competing *Patterns* without taking some liberties with the form.
- Repeating the problem in each *Pattern* that provides a solution may confuse readers by giving them a sense of "deja vu" without explaining the cause for it. They may not realize that there are several solutions to choose from and may expend considerable energy trying to figure out how to apply all the solutions simultaneously!
- Having the problem repeated in each *Pattern* that provides a solution increase the effort required to maintain each *Pattern*.
- The problem section is not the only part of the *pattern* that would have to be duplicated. All *patterns* which solve the same problem should include the same set of **forces**, while the **context** determines their relative priority.

Solution:

When several *Patterns* solve the same problem, make this obvious by pointing out to the reader that there are several solutions to this problem. You can capture the common problem and **forces** in one place using *Separate Problem Description*[11] or *Referenced Problem Description*[12]. If you choose to repeat the problem description as described in *Duplicate Problem Description*[13] you should notify the reader that you have done so.

Related Patterns:

Separate Problem Description solves the problem by factoring out the common solution into a separate *Pattern*.

Duplicate Problem Description solves the problem using the brute force method of cloning the problem description into each *Pattern* which provides a solution.

Referenced Problem Description solves the problem by including the problem description in one *Pattern* and referring to it from each other *Pattern* that provides an alternate solution.

Rationale:

It is very easy for readers to become confused if several *Patterns* have similar or identical problem descriptions unless it is pointed out to them that these *Patterns* provide alternative solutions.

Example:

---

[11] Yet to be written.

[12] Yet to be written.

[13] Yet to be written. *Meaningful Metaphor Name* and *Noun Phrase Name* solve the same problem and share a *Duplicate Problem Description*.

Common Problems Highlighted is itself an example of *Separate Problem Description* since it exists primarily to point the reader to the alternative solutions. This is easily recognized by the fact that the solution section merely refers the reader to a number of other *Patterns*; it acts like a "traffic cop."

E.3 *Pattern: Running Example

Problem:

How can you make it easier for the reader to put a *Pattern Language* into practice?

Context:

You are writing a *Pattern Language* that provides step by step instruction on how to do or implement something.

Forces:

- The *Pattern* should be clear and complete so that reader can use it with minimum effort or chance of mistake.
- The *Pattern* should be as concise as possible without being too terse for most people to understand..
- Many people find abstract descriptions very hard to understand.
- Examples are very useful but must not take a lot of effort or prior knowledge to understand.
- Any one example may not be ideal for explaining a specific pattern.
- When a language contains a significant number of *Patterns*, each *Pattern* must necessarily be more concise than a free-standing *Pattern* if only for reasons of overall *Pattern Language* size.

Solution:

Try to use a single example in all *Patterns* in the language. Explain it once, possibly in the language introduction. Use it to illustrate how each *Pattern* in the language contributes to the solution. Use additional examples where the *Running Example* does not illustrate the pattern effectively.

Rationale:

A single *Running Example* gives the reader more insight into applying the whole *Pattern Language* than a bunch of individual examples. In effect, it is a case study. The reader does not need to invest time and effort into understanding the example for each *Pattern*; they pay this cost only once, when the example is introduced.

Example:

This *Pattern Language* attempts to be a *Running Example* of all the patterns it contains. The authors have also tried to identify one or two examples of each *Pattern* from published pattern works to augment the *Running Example*.

E.4 Pattern: (Distinctive) Headings Convey Structure

Problem:

How do you help the reader understand how the individual *Patterns* he/she is reading fits within the overall structure of the language?

Context

You are writing a *Pattern Language* that has a non-trivial structure. You are applying *Visible Language Structure*[14] because you recognize that it is important for the reader to be able to understand how the individual *Patterns* he/she is reading fits within the overall structure of the language. You are attempting to make the resulting language *Single-Pass Readable*.

Forces:

- In *Pattern Language*s with complex structure, readers may find it easier to appreciate the individual *Patterns* if they understand how they fit within the structure of the language.
- When reading through a complex *Pattern Language* for the first time, it is easy to lose track of where you are within the structure of the language.
- A lengthy "you are here" section for each *Pattern* is repetitive, adds unnecessary bulk to the language, and may distract the reader from the *Patterns* themselves.
- An introductory section, at the beginning of the *Pattern Language*, can be an effective means to communicate the overall structure of the language.
- A language takes longer to read (and is not Single-Pass Readable) if the readers need to constantly refer back to the introductory section to figure out where they are.

Solution:

Make individual *Pattern* headings visibly different from all other document section headings. Prefix *Pattern* headings with hierarchical section numbers, where the section numbering hierarchy parallels the language structure.

Rationale:

The reader can easily recognize *Pattern* sections and can tell at a glance how the given *Pattern* fits within the language structure. Section numbers are concise, and do not distract the reader from the *Patterns* themselves. Major changes in section numbers can signal the reader that he/she has come to a new section of the *Pattern Language*.

Example:

In this *Pattern Language*, we have organized the *Patterns* into 5 major categories lettered A thru E. Within a category any *Patterns* which are clearly subservient to another *Pattern* have been numbered by adding a .1 to the *Pattern* number of the higher level *Pattern*. *Code Samples* (D.2) is a *Pattern* within the *Maximizing Understanding* category (Section D.) *Code Samples as Bonus* (D.2.1) is an extension of Code Samples hence the subordinate numbering. Similarly, *Evocative Pattern Name* (C.3) is supported by *Noun Phrase Name* (C.3.1) and *Meaningful Metaphor Name* C.3.2); all fit within the *Pattern Naming and Referencing* category (Section C.)

In *Episodes* [Ward96], the *Patterns* are divided into three sections, Product, Development, and Programming. The *Patterns* in each section are numbered accordingly.

Related Patterns:

---

[14] Yet to be written. *Visible Langauge Structure*: Ensure that the structure of your language is visible so that the reader can keep track of where they are in the language.

Section-Name Running Footers/Headers is another pattern which can be used to achieve *Visible Language Structure*.

Page Numbered Pattern References make it easier for a reader to flip to the page describing a specific, referenced *Pattern*.

E.5 Pattern: Glossary

Problem:

How do you clarify unfamiliar terminology in a *Pattern Language* without interrupting the flow of the *Pattern*?

Context:

You are writing a *Pattern Language* that involves terminology that may not be familiar to the *Target Audience*.

Forces:

- *Patterns* may need to use terminology that is unfamiliar to readers.
- *Patterns* should be concise. Defining all terms within the *Pattern* description may make it hard to follow for those familiar with the terminology..
- Expanding the terminology elsewhere may require the reader to flip pages often.
- Putting all the definitions in one place makes it easier to find them.

Solution:

Provide a glossary of terms as part of the *Pattern Language*. The glossary gathers terms that are used in multiple *Patterns* within the language with definitions of the terms. If you feel that it is essential to have the definition handy, you may include a short definition of the term in a footnote as well.

Rationale:

A glossary collects terminology from multiple *Patterns* in one place, thereby making the *Patterns* more concise. The definitions make the *Patterns* understandable by people unfamiliar with the terminology. Glossaries are a proven technique used in many written publications to achieve the same purpose.

3.0 Back Matter

Concluding Remarks

This *Pattern Language* is by no means complete. As long as the art of *Pattern* writing continues to evolve and mature, this language will need to evolve with it. There are many areas that this language has not even attempted to cover. As a generative language, it does not prescribe a process for the creating a *Pattern* or *Pattern Language*. A number of such *Patterns* come to mind, *Patterns* such as *Record the Solution*, *Determine the Problem*, *Find the Forces* and *Separate the Problem from the Context*.

Except for the section on *Pattern* Naming and Referencing, this language has deliberately tried to avoid any questions of style. Style is a very personal thing and it is too early in the life cycle of the Pattern to prescribe a specific style. Each style of *Pattern* writing probably warrants its own *Pattern Language*.

The authors hope that you find this language useful in your *Pattern* and *Pattern Language* writing endeavors and that you will share your favorite *Pattern* writing *Patterns* with the *Patterns* community. Please forward any comments to the authors via e-mail.

Acknowledgements

The authors would like to thank all the participants of PLoP-95 for their contributions. Special thanks go to Linda Rising and Brandon Goldfedder who provided feedback on early versions of the language, and to John Vlissides who's "shepherding" of this paper help us get it into the form you see now.

4.0 Appendices

4.1 Problem/Solution Summaries

The following tables summarize the *Patterns* in this *Pattern Language* for reference purposes.

Table 1: Pattern Structure Patterns

Problem	**Solution**	Pattern **Name**
How do you make sure that all necessary information is covered in a *Pattern*	Include the following elements: Pattern Name, Problem, Context, Forces, and Solution	**Mandatory Elements Present**
How do you communicate essential information that does not fit well into the Mandatory Elements?	Include the following sections when they help convey the information: Resulting Context, Related Patterns, Examples, Code Samples, Rationale, and Aliases.	**Optional Elements When Helpful**
How do you ensure that the reader understands the choice of solution?	Regardless of the style chosen for the *Pattern* description, ensure that the forces are highly visible	**Visible Forces**
How do you make it easy to get the essence of a *Pattern* solution quickly?	Write the *Pattern* so that it is not necessary to read the later parts in order to understand the earlier parts.	**Single-Pass Readable**
How do you minimize the amount of reading required to get the essence of a *Pattern*?	Clearly identify the Problem, Context and Solution parts so that the reader can quickly determine whether this *Pattern* applies to them.	**Skippable Sections**

Table 2: Pattern Naming and Referencing Patterns

Problem	**Solution**	Pattern **Name**
How do you refer to other *Patterns* within the description of your *Pattern*?	When referring to *Patterns* within the body of your *Pattern*, weave the *Pattern* names into the narrative.	**Readable References to** ***Patterns***
How do you name a *Pattern* so that it is easily remembered and referred to?	Choose a *Pattern* name that conjures up images which convey the essence of the *Pattern* solution.	**Evocative** ***Pattern*** **Name**
How do you give a *Pattern* a useful and memorable name?	Name the *Pattern* after the result it creates.	**Noun Phrase Name**
How do you give a *Pattern* a memorable name?	Find a meaningful metaphor and name the *Pattern* after it.	**Meaningful Metaphor Name**
How do you make a *Pattern* part of a larger group of *Patterns*?	Read other *Pattern Languages* and describe the relationships to other *Patterns*.	**Relationship to Other** ***Patterns***
How do you refer to external *Patterns* in a concise but meaningful manner, so that the understanding of your *Pattern* is not compromised?	Include the Evocative *Pattern* Name an author/year tag and a footnote with a brief summary of the *Pattern*.	**External** ***Pattern*** **Thumbnails**

Table 3: Patterns for making Patterns Understandable:

Problem	**Solution**	Pattern **Name**
How can you make a software *Pattern* sufficiently clear and unambiguous to facilitate straightforward implementation?	Provide one or more implementation code examples, written in a prevalent programming language, to illustrate the *Pattern* concepts.	**Code Samples**
How can you ensure that the essence of your software *Pattern* can be understood by your entire target audience, regardless of their familiarity with specific programming languages.	Ensure that the *Pattern* can stand on its own, able to communicate its essential concepts even if the code examples were deleted.	**Code Samples as Bonus**
How do you ensure that diagrams are easily understood by your entire target audience?	Use diagramming notations that are likely to be familiar to the target audience.	**Understood Notations**
How do you ensure that a *Pattern* is easily understood by its intended audience?	Identify a clear target audience and keep this audience in mind while writing the *Pattern*.	**Clear Target Audience**
How do you maximize the likelihood of the intended reader understanding your *Pattern*?	Use only those terms with which the typical member of the audience could reasonably be expected to be comfortable	**Terminology Tailored to Audience**

Table 4: Language Structure Patterns

Problem	**Solution**	Pattern **Name**
How do you give the reader an overview of a set of *Patterns*?	Summarize the Pattern Language in the Introduction.	*Pattern Language Summary*
How do you make it easy for a reader to pick out useful *Patterns* that solve their problem?	Provide a table that summarizes all of the *Patterns,* including a brief description of each *Pattern*'s problem and the corresponding solution.	*Problem/Solution Summary*
How do you make readers aware that they should choose one of the alternative solutions?	When several *Patterns* solve the same problem, make this obvious by pointing out to the reader that there are several solutions to this problem.	*Common Problems Highlighted*
How do you help the reader understand how the individual *Patterns* he/she is reading fits within the overall structure of the language?	Prefix *Pattern* headings with hierarchical section numbers, where the section numbering hierarchy parallels the language structure.	*Headings Convey Structure*
How can you make it easier for the reader to put a *Pattern Language* into practice?	Try to use a single example in all *Patterns* in the language.	*Running Example*,
How do you clarify unfamiliar terminology in a *Pattern Language* without interrupting the flow of the *Pattern*?	Provide a Glossary of all terms which may be unfamiliar to the audience.	*Glossary*

4.2 References

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides "*Design Patterns: Elements of Reusable Object-Oriented Software.*" Addison-Wesley. ISBN 0-201-63361-2.

[PLoP94] Proceedings of PLoP-94 - "Pattern Languages of Program Design" published by Addison-Wesley in 1995.

[PLoP95] Proceedings of PLoP-95 - "Pattern Languages of Program Design" published by Addison-Wesley in 1996.

[APLxx] Alexander et al, "A *Pattern Language*", Oxford Press

[Bercz96] Stephen P. Berczuk, "A *Pattern Language* for Ground Processing of Science Satellite Telemetry" in [PLoP95]

[Ward96] Ward Cunningham, "Episodes: A *Pattern Language* of Competitive Development", in [PLoP95]

[Cope96] Jim Coplien, "A White Paper on *Patterns*", SIGS books

[Beck96] Kent Beck, *Smalltalk Best Practices*, to be published by ??? in 1996