Use Cases for Pedagogical Pattern Systems

Michael Derntl
Research Lab for Educational Technologies, University of Vienna
Rathausstrasse 19/9, A-1010 Vienna, Austria
michael.derntl@univie.ac.at

Luca Botturi
NewMinE Lab, University of Lugano
Via Giuseppe Buffi 13, CH-6900 Lugano, Switzerland
luca.botturi@lu.unisi.ch

(Technical Report #0602
Research Lab for Educational Technologies
University of Vienna
January 2006)

Abstract
Coming from Architecture, through Computer Science, pattern-based design spread into other disciplines and is nowadays recognized as a powerful way of capturing and reusing effective design practice. However, current pedagogical pattern approaches lack widespread adoption, both by users and authors, and are still limited to individual initiatives. This paper wants to contribute to creating a shared understanding of what a pattern system is by defining the key terms. Moreover, the paper builds upon and extends a set of existing functional and non-functional requirements for pattern systems, adds structure to these requirements, and derives essential use cases following a goal-based approach for both pattern maintenance and pattern application. Finally, implications concerning the pedagogical use of pattern-based design are drawn, concluding that a stronger focus on the underlying (pedagogical) value system is required in order to make a patterns system a meaningful tool for effective educational design.
1 Introduction

Coming from Architecture first, and then through Computer Science, the idea of pattern-based design has come to education and e-learning, generating the concept of pedagogical patterns. As any innovation, design patterns need to show the specific features that make them appealing for the potential user community [1]. In the software design community the position of design patterns is solid, while in pedagogy there is recognition, yet no broad acceptance. In this paper, we try to point out essential features and requirements of pattern systems, along with other features that might present criticalities. In order to support a sound integration of patterns in education, the research presented in this paper aims at the development of use cases for pedagogical patterns. Generally, use cases are employed to describe interactions between a system and its users, focusing on user goals and involved functional and non-functional requirements. In this respect, we propose essential use cases for pattern systems by following and expanding the “trail of requirements” traced by [2]. As a prerequisite we first elaborate on the concepts and main ideas underlying the notions of pattern, pattern language, and pattern system – the main concepts used for creating a conceptual framework about pattern-based design. We show that there are some basic use cases which are essential to any pattern system. To investigate the use of patterns particularly in the field of pedagogical design, we pose and explore relevant implications of the use cases for a pedagogical pattern language.

A structured and focused approach to design patterns might be useful both to those creating (pedagogical) patterns, and to potential pattern users, to actually see what is possible to achieve with patterns. The use cases and functional requirements can in fact support the analysis and assessment of the value and potential of a pattern language or system in education design. In short, we try to answer two main questions:

- What are the relevant features that a pattern system should present in order to be useful and actually usable by designers?
- What are the essential use cases that authors of patterns and pattern systems can take as relevant for their work?

The next section is devoted to the definition of the main concepts and terms used throughout the paper through literature analysis. They are namely, pattern, pattern language, and pattern system. Section 3 introduces the functional and non-functional requirements for pattern systems, on which, in section 4, we build use cases following a goal-oriented approach. Finally, the last section brings forth some discussion and open issues.

2 Main Concepts about Patterns

To achieve a shared understanding of essential tasks in creating and applying patterns and pattern languages we first need to build a common understanding
of involved terms and concepts. The basic distinctions addressed can be coarsely stated in three sentences:

1. A design pattern conveys the core of the solution to some relevant, recurring design problem [3].

2. A pattern language incorporates a set of patterns relevant to a specific design domain, including rules to combine them (i.e., like words and grammar rules).

3. A pattern language constitutes a pattern system, i.e., an abstract tool that supports users in applying patterns or creating new ones from their experience.

Readers familiar with the pattern approach are no doubt aware of the first two concepts. However, the concept of a pattern system requires more accurate consideration, as we subsequently want to identify system-relevant use cases.

2.1 (Design) Pattern

The term pattern (synonymously used with design pattern) can be used with two different meanings:

1. In a philosophical sense a pattern is a recurring phenomenon in some real or artificial system or artifact. For example, bird flocks have particular flying and migration patterns. In this sense a pattern is real, yet intangible; somehow, it only exists in the mind of the observer, and becomes invisible if observing only one element in the system or artifact (e.g., a single bird). Our interest is for patterns related to artifacts, that is, design patterns.

2. To make the pattern as such explicit and to make it include design advice we have to create a formalized representation using some natural or artificial language. This constitutes the predominant usage of the term pattern (at least in this paper), namely a formalized, explicit representation of the pattern including guidance on why and how to apply it to real designs. For example, a simple architectural pattern might be: if a kitchen is used by more than one cook, placing the fires in the middle instead of along a wall creates more room for cooperation.

2.2 Pattern Language

This term was coined by architect C. Alexander [3]. It describes a (finite) set of patterns, which enables the creation of an “infinite” number of combinations. To support this, patterns in a language must not stand isolated from each other. Instead, they must complement one another in a synergetic way, and at the same time provide instructions and guidance on possible combinations. Alexander built an analogy with natural languages [4, p.187]: Patterns, connections between patterns, and constructs in pattern languages are the equivalents of
words, grammar, and sentences in natural languages, respectively. Over time a number of alternative terms for a set of patterns emerged, some with slightly different meanings or focus: pattern catalog as a collection of loosely related patterns [5], pattern handbook as a summary of relevant concepts in a domain [6], or pattern system as a pattern collection that ties its patterns more together than a pattern catalog [7].

Note that in this paper we will sometimes use pattern collection instead of pattern language when we refer to a set of patterns without any additional assumptions about structure or relationships among them.

2.3 Pattern System

A pattern language provides the users with the tools for its standard use, i.e., applying the patterns. We have standard use of a language when we understand and speak it. A pattern system offers something more: it offers tools for creative use, i.e., a way for creating new patterns by understanding the very roots of the system itself – its meta-language features. In natural languages this is typical of native speakers: their competence with their mother tongue allows them to create new yet understandable words and sentences. In this paper, a pattern system refers to a conceptual system, which consists of the pattern language and some formulation of meta-language features, e.g., instructions how to use the patterns, the underlying value system, philosophical background, as well as other relevant information and requirements (see following section). In a broader sense we could also refer to it as a pattern culture, which includes not only the patterns and their connections as tangible components, but also tacit components like personal values, social aspects, a deeper meaning.

It should be emphasized that a pattern system is a conceptual tool and does not need any computerized interface, but can live on paper. Of course, a digital interface to a pattern system might be a powerful tool for supporting its use.

3 Requirements for Pattern Systems

In this section we lay out the requirements that make pattern systems useful and usable. What (meta-language) features should be visible in the pattern system in order to make it meaningful and useful to its potential users? Thereby we build on Fincher’s selection of functional [2] and non-functional [8] requirements for patterns and pattern languages, and allow ourselves to add arguments based on our own experience and on additional literature. In Fincher’s papers, the requirements are laid out in no particular order. As depicted in Table 1, we added a structuring dimension related to the scope that a requirement addresses in the pattern system. The scope either refers to single-pattern features, pattern language features, or meta-language features.
Tab. 1: Features and requirements for pattern systems. (* indicates requirements added by the authors).

<table>
<thead>
<tr>
<th></th>
<th>Functional requirements</th>
<th>Non-functional requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single pattern</strong></td>
<td>Capture of practice ↔</td>
<td>Non-obvious</td>
</tr>
<tr>
<td></td>
<td>Abstraction ↔ Insight</td>
<td></td>
</tr>
<tr>
<td><strong>Pattern language</strong></td>
<td>Structuring and compositibilty ↔ Generativity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Presentation form ↔ Communicative power</td>
<td></td>
</tr>
<tr>
<td><strong>Meta-language</strong></td>
<td>Value system ↔ Value focus*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Structuring rationale* ↔ Liveliness*</td>
<td></td>
</tr>
</tbody>
</table>

3.1 Single-Pattern Requirements

**Capture of practice.** A pattern must capture a real problem-solution pair in a specific environment. Just writing down the problem and its solution in pattern form is not enough. In the capturing process, the author should give prominence to the idea behind the solution, which is actually the gist of the pattern, its essential quality, which is commonly referred to as the “quality without a name” [2, p.337-8]. This requirement also relates to Lea’s [10, p.42] concept of encapsulation, holding that a pattern has to encapsulate some well-defined problem and solution.

Why do pattern users need more than just the guidelines for a solution? The fact is that applying a high-quality pattern does not automatically lead to high-quality design. The quality of a pattern-based design solution does not depend only on the subjective or objective quality of the patterns used, but equally on the skills and experiences of those who apply the patterns to their designs. Patterns do not automate any design solution: the human factor in pattern use is essential, and the expression of the idea behind the pattern, the sparkle that generated it, is paramount.

For pedagogical design remains the question of which quality we want to capture as “the idea.” The answer to this question lies in the pedagogical and didactical baseline of the target learning and teaching method. For example there is probably no place for a drill-and-practice pattern in a pattern language for constructivist learning environments. Drill-and-practice aims to make individuals adopt some behavior without much reflection, while it conveys none of the essential requirements defined by constructivist learning theory, e.g. facilitating self-directed learning ways. This is why one of our basic claims is that a pedagogical pattern collection needs to be based on some clearly expressed pedagogical baseline. Only then will it be possible to assess a pattern’s quality in terms of the goals and qualities

---

1 While for many the concept of this unnamed quality is lacking scientific foundation or tangibility, it can be considered as the pattern author’s attempt to write a pattern that is capable of imparting to its readers a feeling of connectedness with the design, trying to transfer the expert’s connectedness to the reader [9].
Abstraction. Each pattern is required to capture successful practice, whereas the characteristics of “success” have to be abstracted from concrete examples [2, p.338]. In Alexander’s argumentation, abstraction is addressed by referring to the “system of forces” leading to a specific problem [4, p.247]. Only when understanding the forces at work in a problem’s context, the pattern author can try to find a way of describing a configuration or intervention that is capable of resolving these forces, instead of describing a concrete solution to a concrete problem. Abstraction is emphasized by many works in the pattern community. Notably among them, Riehle and Zllighoven [6] explicitly include the term abstraction into their notion of a pattern by maintaining that a pattern is “an abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts” (p. 3). Similarly, Lea [10, p.42] establishes abstraction as one of the six key features of a pattern, holding that a pattern has to represent an abstraction of some real, empirical experience and everyday knowledge. Winn and Calder [11, p.60-1], in the domain of software patterns, even claim that a pattern should include design information on multiple levels of abstraction. This should on the one hand help designers make connections between different design levels, and on the other hand facilitate the design process by guiding the designer from a general understanding of a problem to its more detailed facets.

It is accepted among many pattern authors that a pattern must be an abstraction based on at least three concrete examples. However, it is also acknowledged by [2] that in pedagogy this principle is hard to maintain, as most teaching occurs behind closed doors and is therefore practically impossible (and pointless) to be observed by other teachers or pattern authors.

3.2 Pattern Language Requirements

Structuring and compositibility. The previous discussion of abstraction only included the relationship between a pattern and examples of solution to problems. As another dimension of abstraction, the structuring principle addresses the organization of patterns within the pattern language or system. In Alexander’s pattern language the patterns are arranged on three different scales, which are reflected by the sub-title of the book, i.e., “Towns, buildings, construction” [3]. This kind of hierarchical relation of more coarse grained patterns with more fine grained ones is also referred to as “compositibility” [10, p.42], hinting to one of the primary intents of structuring: the structuring principle is a vehicle of supporting the stepwise composition of a design artifact. It also makes the pattern collection easier accessible to pattern users by providing (a) entry points for
searching patterns at the appropriate level and (b) a means of navigating through the patterns using their relationships. Interestingly, the structuring principles vary significantly among different pattern collections, as the following (incomplete) list suggests:

- As already mentioned, Alexander’s patterns are arranged on three different levels: *Towns*, the top level, comprises 94 global or large patterns, describing the layout of whole towns and communities; *Buildings*, the middle level, consists of 110 patterns which can be applied to groups of buildings and individual buildings; and *Construction*, the bottom level, comprises 49 patterns showing in detail how to construct the buildings and parts thereof. Along a similar dimension, the POSA (*Pattern-oriented software architecture*) system of patterns [7] differentiates three pattern categories: architectural patterns in the large, design patterns in the middle, and coding idioms in the small.

- The “Group of Four” (GoF, [5]) organize their 23 software design patterns into two dimensions: scope (class or object) and purpose (creational, structural, or behavioral). Additionally, they provide a directed graph depicting how patterns include or depend on each other.

- The PPP [12] authors do not explicitly categorize their patterns. Instead, there are different options of browsing through patterns: by subject, by learning objective, by teaching/learning element, by name, and by author. Some sub-collections of the PPP offer pattern maps showing how different patterns relate to each other, and most of them offer “quick access tables,” relating common problems to suitable patterns. However, the list of problems seems to be unstructured.

- The E-LEN project [12] authors organize their patterns according to the four special interest groups (SIG) in which they were produced: (1) Learning resources and learning management systems, (2) Lifelong learning, (3) Collaborative learning, and (4) Adaptive learning. Only the first SIG uses additional categorization of patterns into access patterns, learning patterns, instructional patterns, and informational patterns.

**Presentation form.** Patterns within a pattern language are usually presented uniformly. Most patterns contain problem and solution statements, but the other parts may differ among different approaches [2, p.342]. For example, Alexander uses a rather prose-style template comprising a picture and a number of formatted paragraphs including problem, solution, forces, and connections to other patterns. The GoF patterns, in contrast, expose a different set of about a dozen sections (e.g., intent, motivation, implementation, etc.), each explicitly referred to and provided in the pattern
document. There are many other pattern forms, but the Alexandrian form is prevalent today. We conclude that a uniform presentation among patterns in a domain or collection is reasonable, because it supports reader orientation (i.e., quickly finding desired information) as well as comparison of related patterns (e.g., comparison of relevant problem statements).

3.3 Meta-Language Requirements

Value system. Design is a value-laden activity, and good design must encompass values of interest to the target community [2, p.340]. This is supported by [13], who claim that a pattern language for design has to be explicitly committed to a set of values which characterize well-being in the environment created through applying its patterns (p.451). Others refer to the key values as the “leitmotiv” of the patterns [6].

For pedagogical design this means that it makes little sense to create a pedagogically neutral pattern language, as it could not be assessed by potential users. On the other hand, it makes sense to create a pedagogically neutral notation language for patterns, i.e., a language that can express in a standard and comparable format patterns referring to different values.

Structuring rationale. This requirement was not included in Fincher’s list. We added it to reflect the need for a specification of the rationale behind the structuring and organization principles in a pattern system. If this rationale is explicitly stated and transparent, the actual structure and the patterns become a “lively” system, which is better capable of adapting to emerging or changing external requirements. For example, we not only say that there are three layers (as in Alexander’s language), but we also state why we allocate the patterns to these three layers.

3.4 Non-Functional Requirements

In addition to the key functional requirements, Fincher and Utting [8, p.200-1] later added a handful of essential non-functional requirements (NFR), each paired with one of the functional requirements. We believe that these NFRs equally deserve to be mentioned, as they complement the functional requirements with those located more on the user’s side, contributing towards the “quality without a name.”

- **Non-obvious.** (Paired with capture of practice.) This NFR requires that a pattern must capture practice which is non-obvious. This can be achieved when a pattern emphasizes on why to do something instead of focusing solely on how to do it.

- **Insight.** (Paired with abstraction.) A pattern should convey design insights beyond details of implementation, namely “distilled” expertise that shows why the pattern is appropriate for a given problem. To underline what is needed for distilling insight, we take a look at the characteristics
of *expertise*: Applying good and bad solutions to various kinds of problems and reflecting upon the results is what can make an expert out of a novice in some domain, because it contributes to building experience. Among other factors like domain-specific knowledge, it is the experience which experts have internalized, and which makes them capable of solving complex problems by relating new situations to situations previously successfully solved. A good pattern distills and explicates such experience-based, tacit insight.

- **Generativity.** (Paired with *structuring principle.*) Generative patterns are active and dynamic, facilitating the generation of large designs composed of smaller pieces, as opposed to non-generative patterns, which describe “recurring phenomena without necessarily saying how to reproduce them” [9]. Application of a pattern usually triggers the creation of a new problem context in which one might have to solve a different problem by applying additional patterns. This process is best supported by generative patterns, exposing various forms of relationships among each other.

- **Communicative power.** (Paired with *presentation form.*) The pattern format is intended to communicate a pattern’s essence. Important parts in this respect include the pattern name, which has to be chosen carefully, as it will be used as shorthand form for the whole pattern. Additionally, images, models, and illustrative examples seem very appropriate to support a pattern’s information.

- **Value focus.** Note that this NFR is not included in Fincher’s list. We added it to pair up with the *value system* requirement: we consider it important that the essential qualities of the underlying value system shine through to the pattern user or reader. This should contribute to easier understanding on the side of the users, because they can (implicitly or explicitly) relate a pattern’s essence to some value that is (explicitly) underlying the whole pattern collection. Explicit value focus narrows the expectations of users to relevant designs only (given they are familiar with the value system).

- **Liveliness.** This NFR, which we also added to Fincher’s original list, accounts for the non-functional aspect of the *structuring rationale*. The structuring and organization principles of the pattern system should be created such that they reflect liveliness, in the sense of openness to change, adaptability and extensibility. Even though Alexander was aiming his patterns towards a “timeless way” of building, we claim that a pattern language should not be considered complete at any time, as the real world it captures is constantly subject to change. So it could be wise to account for adaptations and extensions not only to the set of patterns in a language, but equally to their structuring principle.
4 Use Cases for Pattern Systems

Use cases are means of documenting what a system should do [14]. Their primary intents are (a) specifying the functionality and requirements of software systems in collaboration with involved stakeholders and (b) providing easy-to-understand models and descriptions of what the system should achieve for its users. Even though use case modeling is mostly used in software projects, it is equally valid to model use cases for other systems, as long as the respective system creates value for its users, in collaboration with its users.

4.1 Approach

For eliciting and structuring pattern system use cases, we basically follow the steps involved in a goal-driven approach (cf. [15]). This seems reasonable, as we already hinted to the primary goals and requirements.

We employ the approach in a simplified form, while adhering to its three basic steps:

1. Identify actors: This is done by investigating types of users directly interacting with the system. We identified four relevant actor roles: An author has to write the patterns, a maintainer has to maintain the resulting pattern system, an analyst uses the patterns to analyze design artifacts, and all these are users of the patterns, including those who just apply them. Both authors and maintainers are users, and an author somehow also maintains the pattern system. An analyst is a specialized user. This yields the actor hierarchy given in Figure 1.

2. Identify goals: In addition to the requirements presented in the previous section, we want the actors to be able to achieve the following main goals in a pattern system, irrespective of the domain in which the pattern are located:

   \( G_1 \): Capture successful practice
   \( G_2 \): Improve design artifacts and experience

Fig. 1: Actors hierarchy in pattern systems.
4 Use Cases for Pattern Systems

$G_3$: Advance design knowledge in a domain

$G_4$: Keep an up-to-date “knowledge base” of successful practices

$G_5$: Describe and structure regularities among desirable practices

$G_6$: Reuse successful practices

$G_7$: Acquire overview and specific insights in the domain

$G_8$: Communicate with peers about design solutions

$G_9$: Understand why certain techniques and practices (don’t) work

$G_{10}$: Analyze a set of concepts (artifact) with respect to relevant factors

The list can certainly be extended, especially when considering goals specific to particular domains.

3. **Build use-case models**: This step is quite straightforward, as we just have to derive the essential use cases that support the main user goals defined in step 2.

By analyzing the user goals we identify the following basic supporting use cases:

- **Write Pattern**, supporting $G_1$ to $G_5$.
- **Revise Pattern**, also supporting $G_1$ to $G_5$.
- **Maintain Patterns**, an abstract use case describing the shared intent of the previous three use cases, primarily covering $G_1$ to $G_5$.
- **Identify Patterns**, supporting $G_1$, $G_3$, and $G_{10}$.
- **Apply Patterns**, supporting $G_2$, $G_6$, and $G_8$.
- **Browse Patterns**, supporting $G_7$ and $G_9$.
- **Analyze an Artifact**, supporting $G_3$, $G_7$, $G_9$, and $G_{10}$.

To draw a clear separation between the maintenance side and the application side of a pattern system, we organize the use cases into two different packages: “pattern maintenance” (see Section 4.3) and “pattern application” (see Section 4.4). This should also contribute to keep the models clear [16].

4.2 **Use Case Template**

There is no agreed-upon template for use case descriptions. The use case template considered appropriate to describe the abstract “services” offered by pattern systems includes the following information:

**Use case name**: A few words giving the use case a succinct name to refer to.
Summary: A short summary conveying the essence of the use case events, i.e. what the use case achieves for the user. Each use case should include some objective or goal to reflect the value created for the user.

Actors: Actors involved in the use case.

Preconditions: Conditions which have to be satisfied before an instance of the use case can start.

Basic course of events: A course of events covered by the use case showing how the pattern system interacts with the external actors. Note that these events are less formal and strict than use case events in software systems, as a pattern system (as a conceptual system) is not really capable of “responding” to user interactions like a software system does. Further note that use cases for software systems are usually complemented by alternative courses of events [17], which is mainly due to the highly interactive nature and error-proneness of software systems. For pattern systems, we conjecture that specifying alternative event flows would not add much value to the use case description.

Postconditions: Conditions which have to be satisfied when the course of events is completed.

Relevant requirements: This section explicitly refers to the functional and non-functional pattern system requirements addressed or affected by the use case (and vice versa).

Related use cases: As a complement to the graphical notation of relationships among use cases (including those too informal to be depicted in the diagrams), this section provides a short textual description of relationships that this use case maintains with others.

Remarks: Optional concluding remarks and annotations to any of the use case's information.

4.3 Package “Pattern System Maintenance”

This package includes use cases relevant to all aspects of pattern system maintenance, including identifying patterns, writing them down, refining them, etc. (see Figure 2). The users primarily involved in these use cases are the pattern maintainer and the author.

4.3.1 Use Case: “Maintain Patterns”

Summary
This is an abstract use case, acting as the base container of events and conditions concerned with modifying the contents of the pattern language to reflect new or changing circumstances. This allows pattern maintainers
to create, update, and remove patterns. Each use case deriving from this one must clearly address one single maintenance action.

**Actors**
- Maintainer

**Preconditions**
- Value system defined: The value system underlying the pattern language is the most important prerequisite for pattern maintenance. Only when the value system is clearly and explicitly defined is it possible to answer questions such as: Is it justifiable to add a new pattern due to new observations? Is a pattern outdated and has to be removed? Does a pattern need revision?
- Structuring principles defined: The structuring principles underlying the organization of patterns in the pattern language are defined, and allow for the desired maintenance action.
- There is a clearly stated, justifiable need for the desired maintenance action.

**Basic course of events**
1. *This step is not specified here; it needs to be specified by concrete use cases deriving from this one.*
2. Create history entry to allow for recording changes.

**Postconditions**
• Value system satisfied: The value system underlying the pattern language was not violated by the pattern maintenance action.
• Any navigation maps, indexes, dependent patterns, and other user aids relying on the structuring principles of the pattern language are up-to-date.

Related requirements
• Value system: The primary objective of each maintenance action is to transport and maintain the value system of the pattern language, ensuring the value focus.
• Structuring principles: Pattern maintenance must conform to the structuring principles of the language. This is important for example when pattern structures and dependencies change. Generativity of the pattern language must not be broken by changes.

Related use cases
WRITE PATTERN, REMOVE PATTERN, and REVISE PATTERN are specializations of this use case.

* * *

4.3.2 Use Case: “Write Pattern”

Summary
This use case is a specialization of MAINTAIN PATTERNS. The maintenance objective is the writing or creation of a new pattern by a pattern author.

Actors
Author, (possibly other stakeholders)

 Preconditions
• Inherited: Preconditions inherited from MAINTAIN PATTERNS.
• Examples or arguments: The candidate pattern is justified by either a sufficient number of quality examples, or otherwise by rigorous, convincing argumentation. This precondition heavily depends on the policy defined by the pattern language for adding patterns, e.g., if it adheres to the “rule of three,” three examples or observations of the pattern in the real world are required.
• Contradiction or overlap: There is no other pattern in the pattern language which would contradict or, conversely, overlap with the candidate.
• Presentation form: The presentation form of the pattern language is defined, and all necessary and sufficient information to describe the candidate using this presentation form must be available.
Basic course of events

1. Collect and structure all available, relevant information.
2. Write down the pattern in the defined presentation form.
3. Revise or update patterns which are affected by the new pattern, and update indexes and maps depicting the network of patterns in the language.

Postconditions
Inherited: Postconditions inherited from MAINTAIN PATTERNS.

Related requirements
- **Capture of practice**: The pattern author should capture successful, non-obvious practice.
- **Value system**: The pattern should comply with and carry forth the value focus of the pattern language.
- **Abstraction**: The pattern should provide the user / reader with valuable, reusable insights to the problem environment by abstracting from specific problems or observations.
- **Structuring principles**: The pattern should be inserted into the pattern language according to its structuring principles. This insertion should align with the generative network of patterns in the language.
- **Presentation form**: The pattern should be presented in the defined presentation form, and should embody communicative power through lively writing, a well-chosen name, and illustrative examples.

Related use cases
- This use case is generalized by MAINTAIN PATTERNS
- Writing down the pattern optionally follows identification of a pattern (IDENTIFY PATTERNS)

4.3.3 Use Case: “Remove Pattern”

Summary
A pattern may become outdated over time. Such a pattern needs to be removed when the problem it addresses disappears or when there are better alternatives [7]. This can happen due to changing environmental conditions, or is it also possible that the creation of new patterns and the revision of existing patterns require the removal of a pattern.

Actors
Maintainer
Preconditions

- Inherited: Preconditions inherited from Maintain Patterns.
- One or more of the following:
  - The problem or situation which the pattern addresses disappeared. For example, a new tool becomes available, which makes the solution described by the pattern superfluous.
  - The value system of the pattern language was changed to reflect new findings or environmental conditions.
  - There are better alternatives for solution or pattern available (in this case it might be more appropriate to do a revision of the pattern – see Revise Pattern).

Basic course of events

1. Remove pattern from pattern language.
2. If required, update other patterns to reflect changes (see Revise Pattern).

Postconditions

Inherited: Postconditions inherited from Maintain Patterns.

Related requirements

Value system: The value system (changes therein) can be the reason for the need to remove a pattern.

Related use cases

This use case is generalized by Maintain Patterns.

* * *

4.3.4 Use Case: “Revise Pattern”

Summary

Here, revision refers to a change or update in textual description, intent, or meaning of a pattern. A pattern needs to be updated when the context or the practice it refers to have changed. Also, pattern language internal rearrangements might require the revision of a pattern. For example, desired or needed change in focus and/or abstraction of the pattern; or observing that the value system can be addressed or supported in a better way.

Actors

Author

Preconditions

- Inherited: Preconditions inherited from Maintain Patterns.
• One or more of the following:
  – The pattern can address the underlying value system more adequately.
  – The problem context underlying the pattern has changed. For example, new tools are available to address the problem more efficiently.
  – One or more patterns have been changed and require the change of another pattern. This seems most likely when a new pattern is added.

Basic course of events

1. Collect required material for revision.
2. Do revision by rewriting required sections of the pattern (and ensure that the value system is not violated).
3. Check if this revision requires other patterns to be revised.

Postconditions

Inherited: Postconditions inherited from MAINTAIN PATTERNS.

Related requirements

As pattern revision also includes the writing aspect, this use case affects all functional and non-functional requirements in much the same way as WRITE PATTERN does.

Related use cases

This use case is generalized by MAINTAIN PATTERNS.

* * *

4.3.5 Use Case: “Identify Pattern”

Summary

This use case represents one of the most essential, yet indefinable scenarios in a pattern system: the discovery or identification of a pattern from doing and observing successful practice. In this respect, this is more an “event case” than a use case. Anyway, its main objective is describing the kind of awareness encountered when recognizing a pattern.

Actors

User (actually, any actor can identify a pattern; therefore we link the most abstract actor to this use case)

Preconditions

• Observation or use of design artifacts relevant to the value system of the pattern language.
• The observed design artifact is representative of a successful, purposeful design practice in the respective domain.

• The artifact is not bound to one specific problem only, but can be valuable for similar, yet different problems.

Basic course of events
It is not really reasonable to (pre-)describe an ordered list of events leading to the identification of a pattern, as it can be done – or better, can happen – in many different ways. For example, identification by chance through repeated observation would be the opposite of employing pattern mining techniques for purposeful identification. For the latter the following flow of events is proposed:

1. Do **ANALYZE AN ARTIFACT** for various design artifacts relevant to the pattern language; keep **value system** in mind

2. Extract recurring practices and try to synthesize abstractions of the practices: Are they worth being **captured**? Would they provide valuable **insight** to potential users?

Postconditions
Solid evidence (artifacts, arguments) and support material (examples, literature, etc.) for a new pattern candidate are available.

Related requirements

- **Capture of practice**: Identification should focus on practices that are non-obvious.

- **Value system**: The practices must comply with the **value focus** of the pattern language.

- **Abstraction**: It should be possible to distill valuable **insight** by abstraction from specific practices and artifacts.

- **Structuring principle**: The practice to be captured should align with the **generativity** requirement.

- **Presentation form**: The pattern identified should be expressible in the presentation form.

Related use cases

- **ANALYZE AN ARTIFACT** in the **pattern application** package: During repeated analysis of concrete design artifacts at different levels, pattern candidates may begin to “shine through.” Thus identifying a pattern is not a necessary, but definitely a desired part of analyzing and scrutinizing design artifacts.

- **WRITE PATTERN** is a desirable extension to this use case to capture the practice identified here.
4.4 Package “Pattern Application”

This package includes use cases that are concerned with actually applying and using the pattern language and its patterns (see Figure 3). With respect to the actual design task, it is a rather abstract outline of what can be achieved with patterns. The intent is that in concrete application domains the pattern authors supply more concrete, domain-specific descriptions of how to design artifacts by using their pattern language.

4.4.1 Use Case: “Browse Patterns”

**Summary**

To select a pattern for usage a user must be able to browse through published patterns, like in a “library.” Therefore the system must offer a comfortable, user-friendly way of accessing patterns and navigating through them.

**Actors**

- User

**Preconditions**

- Patterns must be available for browsing.

**Basic course of events**

1. User enters the pattern library, either with a specific pattern in mind or just to browse and read.
2. System presents the user an index of patterns, as well as some default entry point for reading.
3. User selects the entry point.
4 Use Cases for Pattern Systems

4. System presents the desired pattern.
5. From the current pattern, a user may navigate to a pattern referred to by the current pattern, or just return to the index or default entry point.

Postconditions
(None)

Related requirements
Structuring principles: Provides a way of presenting the pattern index, and for providing appropriate default entry points at various levels. The structuring principles of the pattern system should facilitate browsing the patterns. From any specific pattern, the user must be able to navigate to related patterns (supports generativity of patterns).

Related use cases
This use case can be extended by Apply Pattern, if the user finds an appropriate pattern for the design problem at hand.

* * *

4.4.2 Use Case: “Apply Pattern”

Summary
The objective of this use case is to take the selected pattern and apply it to some real problem. This means creating a design artifact in the real world by using (combinations of) patterns.

Actors
User

Preconditions
There is a problem that matches a pattern intent (i.e., one of the patterns is applicable).

Basic course of events
1. User selects a pattern to apply (may include BROWSE PATTERNS).
2. User applies the pattern according by the guidance supplied by the pattern system.

Postconditions
The problem is solved or at least alleviated.

Related requirements
• Capture of practice: A pattern conveying the essence of expert practice instead of abstract arguments is easier to apply.
• **Value system**: A pattern keeping the value focus is more tightly aligned with the users’ expectations and should therefore be easier understandable and applicable.

• **Abstraction**: The pattern will be easier to apply when it embraces expert insight at a level of abstraction that is not bound to a certain problem instance, while giving enough guidance for applying it to the users’ concrete problem at hand.

• **Structuring principle**: As a result of applying a pattern, a new situation or context is created in the target environment. *Generative* patterns hint the user to patterns potentially applicable in the resulting context.

• **Presentation form**: Well presented patterns focusing on essential qualities of the solution should alleviate application of the pattern.

**Related use cases**

**BROWSE PATTERNS**: Especially for users who are new to the pattern system, patterns are found by browsing through it.

**Remarks**

Note that there are different ways of applying patterns, just as there are different ways of structuring and writing them. Pattern collections usually supply custom, stepwise descriptions of how to apply patterns.

For example, Alexandrian patterns are applied through a process called “piecemeal growth,” whereby a problem is solved by successively applying patterns along the hierarchy from higher-level to lower-level patterns. Instantiating a pattern generates a new context which itself may match with the initial context of another pattern. This iterative and incremental process is repeated recursively until either the problem is solved, or until there are no more applicable patterns at hand [3].

The GoF pattern authors recommend a list of seven steps to be taken in using a pattern: (1) Read the pattern once through for an overview. (2) Go back and study the Structure, Participants, and Collaborations sections. (3) Look at the Sample Code section to see a concrete example. (4) Choose names for pattern participants that are meaningful in the application context. (5) Define the classes. (6) Define application-specific names for operations in the pattern. (7) Implement the operations to carry out the responsibilities and collaborations in the pattern. [5, p.29-31]

* * *

**4.4.3 Use Case: “Analyze an Artifact”**

**Summary**

Deals with analyzing and describing existing design artifacts in terms of available patterns. This can produce valuable input for identifying new
patterns (see IDENTIFY PATTERN), for verifying existing patterns by finding supporting examples, as well as for understanding and documenting complex designs.

**Actors**
Analyst

**Preconditions**
Design artifact with relevance to the pattern system is available.

**Basic course of events**
1. Analyst selects relevant artifact.
2. Pattern system offers patterns to analyst (see also BROWSE PATTERNS).
3. Analyst matches patterns with artifact under analysis.

**Postconditions**
(None)

**Related requirements**
- *Capture of practice*: If the relation between the pattern and a concrete practice is evident, the analysis task is supported.
- *Value system*: Finding the pattern systems values in the artifacts should also support the analysis task.
- *Abstraction*: Only abstraction from concrete problems allows analyzing different design artifacts using the same pattern.
- *Structuring principles*: The analysis process requires a generative structure of patterns, although “applied” in the opposite direction: it is needed to “de-generate” complex designs, and to facilitate the analyst in a stepwise decomposition of a complex artifact into smaller, more tangible pieces of design practice. For example, if the design artifact is a university course, the pedagogical pattern language should include pedagogical practice at different scope levels (such as course, module, and unit).
- *Presentation form*: The form should allow the analyst do recognize patterns relevant to a concrete observation in the design artifact.

**Related use cases**
BROWSE PATTERN: Supports the analyst in finding a pattern relating to (a piece of) the artifact under analysis.
5 Implications for a Pedagogical Pattern Language

In the first part of this paper we tried to point out the main features that a pattern system should have in order to become a meaningful tool for the practice of its potential users. In the second part of the paper we tried to express these features in use cases, for a more practical use. What are the implications of this work for a potential pedagogical pattern language?

First of all, it is paramount to determine what (functional and non-functional) requirements a pattern system satisfies, and what use cases it supports. We believe that structuring principle and value system are the most relevant features, as they (a) provide concrete access points to the language for users and (b) are probably the most controversial issues in pedagogical design. Note that this is substantiated by the fact that in the use cases’ sections for related requirements, these two requirements are referenced most often (i.e., seven times each).

Also, we believe that a use case package specifically dedicated to pedagogical patterns and their requirements would need specific rearrangements that experience and application will show. For example, is “designing a course” a specific case of “select and apply patterns”? Can “analyze a course” be understood as “analyze an artifact”? Also, the structuring of the pattern system might vary considering matter-specific issues.

Working towards answers to these questions and towards a (unified) pedagogical pattern language let us take a look at the central implications of the use cases and their requirements. Note that, to link the discussion to current practice, we consider the most prominent pedagogical pattern initiative today, the PPP [12].

- **Maintain Patterns** (representing sub-use cases): These use cases seem rather invariant among different pattern systems and domains. Looking at the PPP patterns and the current discussion, there seems to be implicit consensus that pedagogical patterns should be described in Alexandrian format. Obviously missing in the current maintenance efforts of the PPP is dedicated value focus and consistent use of structuring principles both on language and meta-language levels. From the discussion in this paper we see that these two should be taken into account more seriously to address the needs of potential users and authors.

- **Identify Pattern**: For this use case we have a special situation in pedagogy, as teaching takes place behind closed doors and is hard to observe. Therefore it is important that instructors and learning designers keep notes – or even better, documented models [18] – of their successes and failures working with students.

- **Browse Pattern**: Users must be supported in finding some pattern for special problems (e.g., how to draw attention to a discussion in a seminar?), but also for planning larger instructions (e.g., pedagogical options of planning for a whole laboratory course). For example, it is currently only possible to access PPP patterns through distributed, dislocated web
sites, without any central index or entry point linking uniform resources. As so often, authors just put papers including patterns on the web. While this is convenient for authors and maintainers, the user side is clearly neglected.

- **Apply Pattern**: To provide users with insightful, generative pieces of pedagogical design advice it would be necessary to first think of the primary application use cases in this domain. We propose that these should at least include “Design a course,” and “Design a learning activity.” While in this respect some current PPP efforts succeed in being generative and well structured (e.g. [19]), it seems that others (e.g. [20]) lack particular focus (“What can I expect?”) and clear entry points (“Where to jump in?”).

- **Analyze an Artifact**: Analysis of existing design artifacts is central to pattern discovery and maintenance in any pattern system, hence as well in pedagogical pattern systems. As teaching and learning are (sometimes synchronized) processes, it seems reasonable to employ not only textual documentation techniques like the PPP does, but equally educational / conceptual modeling techniques to capture and document goals, teaching process, learning activities and environment, etc. (e.g., [21, 22, 23]).

As a preliminary conclusion we hold that neither functional nor non-functional requirements are satisfied adequately in current pedagogical pattern efforts. Besides patterns and small pattern collections, the situation does not look any better for a pedagogical pattern language. In this respect we hope that the requirements and use cases reviewed and elaborated in this paper contribute towards the right direction.

6 Conclusion

To conclude and summarize what we found during our research, we go back to the two central questions that were posed in the introduction to this paper:

1. **What are the relevant features that a pattern system should present in order to be useful and actually usable by designers?** The way to a shared conception of the most important features of patterns was laid out years ago by Fincher [2], who proposed a set of basic requirements for patterns. We adopted these requirements, supported them by recent literature, structured them into three conceptual levels (patterns, language, meta-language), and extended them primarily on the meta-language level. The investigation showed that the most salient requirements – and equally the two most missing ones in current practice – are (a) dedication to a consistent, well-defined value system underlying the patterns and (b) provision of lively, generative structuring among patterns. These two, which are both located on the meta-language level, would ensure that users get what they expect from the patterns, and authors have the conceptual
2. What are the essential use cases that authors of patterns and pattern systems can take as relevant for their work? After discussing the most important requirements, we added the main user goals, and adopted a simple approach to deriving the most essential use cases for pattern systems. We found that a pattern system exposes two interfaces to its users: On the maintenance side, the most essential use cases are writing patterns, revising patterns, removing patterns, and identifying patterns. On the application side, users apply patterns and analyze design artifacts, which might require browsing through the patterns.

The main step forward made by the paper is the proposal of a structure for central concepts around patterns, for different dimensions and scopes of requirements, and for essential user interactions within a pattern system. The use cases help to operationalize and structure pattern requirements by showing the basic interactions of pattern users with pattern systems. We provide a conceptual scaffold, which pattern authors and providers can extend and use to construct meaning around essential practices in their domain. Regarding pedagogical patterns, there is yet a long way to go in this direction. Along this way, the approach presented in this paper can be employed to derive and refine custom use cases for patterns of specific pedagogic domains, by building on the base models provided here.

References


