A Pattern Language for Teaching Design Patterns (Part 2)

Christian Köppe
HU University of Applied Sciences Utrecht, Netherlands

Please note: this is the author's version of the work.
Teaching design patterns introduces specific problems like e.g. ensuring that the purpose of patterns is understood and that patterns are applied in the appropriate and correct way. The pattern language for teaching design patterns addresses these problems and offers solutions for teachers and trainers how to solve them.
This second part covers 4 patterns of the language in detail — BEST FITTING PATTERN CHOICE, EXPERIENCED ADVANTAGE, PATTERN IMPLEMENTATION MATTERS, and PRINCIPLE-SUPPORTING PATTERN USAGE — and describes the changes made since the publication of part 1 of the language, published in the proceedings of the EuroPLoP '11 conference. The other patterns are included here as patlets.

Categories and Subject Descriptors: D.1.5 [**Programming Techniques**] Object-oriented Programming; D.2.10 [**Software Engineering**]: Design—*Design Patterns*; K.3.2 [**Computers and Education**]: Computer and Information Science Education —*Computer science education*

General Terms: Design, Languages, Education

Additional Key Words and Phrases: Educational Patterns, Design Patterns

ACM Reference Format:

Köppe, C. 2011. A Pattern Language for Teaching Design Patterns (Part 2). 18th Conference on Pattern Languages of Programs (PLoP) – Portland, Oregon, USA(October 2011), 17 pages.

1. INTRODUCTION

I hear and I forget.
I see and I remember.
I do and I understand.
Confucius

This work is part two of a whole language. An introduction to the language is given in part one and published in the proceedings of the EuroPLoP '11 conference [Köppe 2011a].

This second part introduces new pattern names, which were changed based on feedback from different reviewers. We believe that these names better reflect the core of the patterns and are therefore easier applicable as vocabulary when applying the language. However, the old names are given too — also in the patlets — so that a matching with the patterns presented in part one can easily be made.

We also added a clarification of the sketches to provide the thoughts behind the sketches and their relation to the patterns. One major change was the addition of pattern qualifications, as feedback showed that some of the patterns are

author's address: christian@koeppe.nl

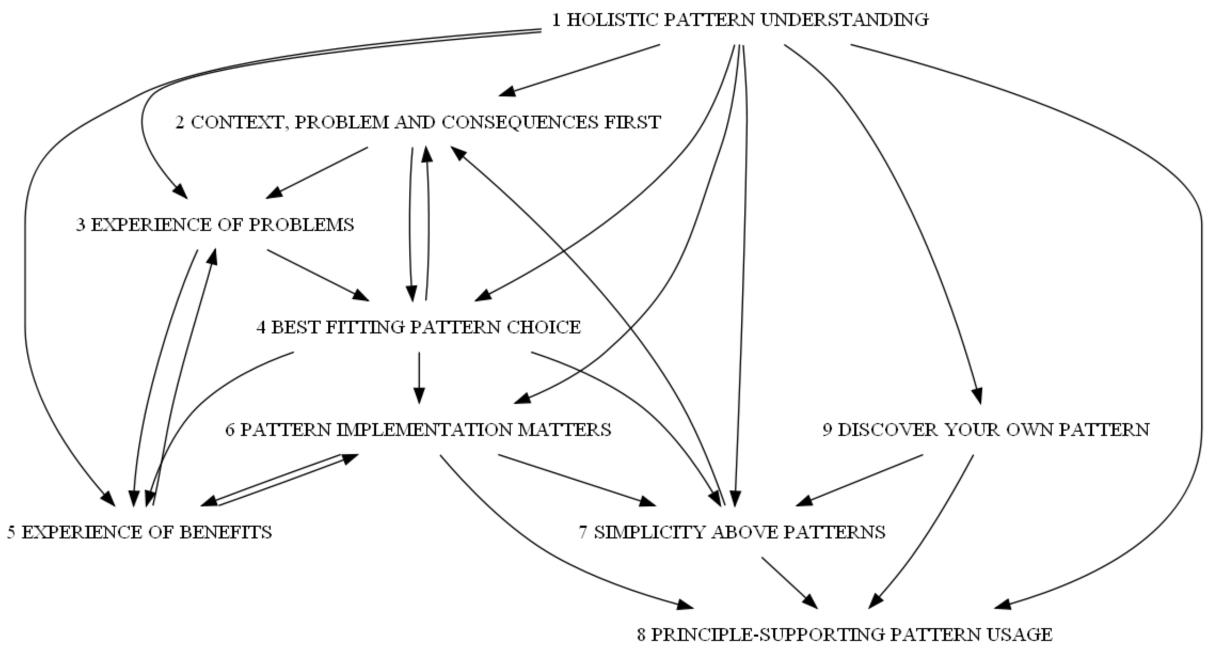

1 HOLISTIC PATTERN UNDERSTANDING
2 CONTEXT, PROBLEM AND CONSEQUENCES FIRST
3 EXPERIENCE OF PROBLEMS
4 BEST FITTING PATTERN CHOICE
6 PATTERN IMPLEMENTATION MATTERS
9 DISCOVER YOUR OWN PATTERN
5 EXPERIENCE OF BENEFITS
7 SIMPLICITY ABOVE PATTERNS
8 PRINCIPLE-SUPPORTING PATTERN USAGE

Fig. 1. Language Map - the arrows show relations between the patterns

also applicable in the broader context of general pattern teaching.

2. THE PATTERN LANGUAGE

Figure 1 provides an overview of the pattern language using the new pattern names, whereby PATTERN UNDERSTANDING serves as entry pattern of the language.

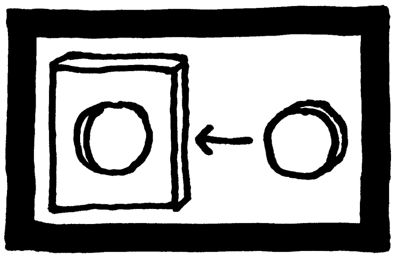
All patterns in this language are just parts of the whole language and should not be used in isolation. Even if the application of separate patterns would add some value relating to the learning objectives, the combination of the patterns will lead to a much deeper understanding in total: the whole is greater than the sum of its parts.

These patterns use a version of the Alexandrian pattern format, as described in [Alexander et al. 1977]. The first part of each pattern contains a sketch, a quote and a short description of the context, followed by three diamonds. The sketch and quote are intended for representing the core of the pattern in recognizable way. However, the ideas behind the sketches are summarized at the end of the paper. In the second part, the problem (in bold), the empirical background, and the forces are described, followed by another three diamonds. The third part offers the solution (again in bold), the qualification of the pattern, the consequences of the pattern application — which are part of the resulting context — and a discussion of possible implementations. In the final part of each pattern in italics, we present some known applications.

Although initially the patterns of this language were intended for application of teaching software design patterns only, it shows that some of the patterns are on a higher level/qualification. These patterns actually form true invariants in the context of teaching patterns of any domain. These patterns are marked — using the notation as introduced by Alexander — with two asterisks. The patterns which are only applicable in the context of teaching (software) design patterns, but form invariants in this domain, are marked with one asterisk. The patterns which are only applicable in specific circumstances and therefore do not represent an invariant, are not marked with an asterisk. A reasoning about the given qualification of each pattern is included in the description of the pattern directly after the solution statement.

Pattern 4: BEST FITTING PATTERN CHOICE**

Also known as: Perfect Fit.



Fine art is that in which the hand, the head, and the heart of man go together.
John Ruskin

You want to make sure that students apply the correct pattern for solving a problem and not just one of the possible pattern solutions.

❖ ❖ ❖

Students often choose inappropriate patterns without exploring if the problem they have is the same as the problem addressed by the pattern. And even if this fits, the context or forces may be different or the consequences are worse than the original problem.

One of the underlying problems is that students have difficulties with determining which pattern to use for a specific problem [Pillay 2010], even if the problem is described properly.

Solution-focussed. Some design patterns describe similar solution structures for solving different problems, like the ADAPTER, PROXY, and FACADE patterns. The differences between these patterns are mainly in their intentions. If not all parts of the pattern are examined, then it might seem that different solutions could be applied to solve the problem at hand. Technically all these solutions could be working, but often the role-names of pattern participants — as suggested in the the GoF-book [Gamma et al. 1995] — are used as parts of the class- or method-names of the concrete pattern implementation. Using the solution of the wrong pattern would therefore communicate the wrong intention.

Vague Problem Description. Some design patterns offer different solutions to similar problems, like the INTERPRETER and COMMAND pattern. If the design problem is — or can — only be vaguely identified, then it's hard to determine which pattern to use, because it can seem that the problem/context-part of more than one pattern matches the problem. It also will be harder to understand the consequences completely.

Big Problem Space. Often there is more than one design problem which needs to be addressed. Choosing a pattern based on just one of these problems could have a negative effect on the other problems.

Example-based Learning. If examples are used which come from books or websites, then these examples often do not state a (sufficient) reasoning of why the applied pattern has been chosen to solve the example problem. This can be because there is not really a problem at hand, which is often the case with e.g. websites which focus on the *implementation* of patterns and not the correct *application*. Students tend to look for examples where patterns were applied in a context similar to their own. But if this example application is based on the wrong decisions or a misunderstood problem, then also the students will apply the possibly wrong pattern.

First Shot Solution. While looking for an applicable pattern, students tend to stop after finding the first possible *solution* to implement. So they are using “first possible solution" as stop condition for their search, which is faster than determining *all* possible patterns, studying their descriptions and making a well-grounded choice of one these patterns. Even if this is technically working, it could communicate a wrong intention.

❖ ❖ ❖

Therefore: Ensure that the students have analysed the design problem, context, forces, and consequences sufficiently, and that all match with the pattern they choose. It is also important that the resulting context and the fitting to other applied patterns has been taken into account.

This pattern is a true invariant. Independent of the domain, problems can look similar in different contexts or different solutions exist for the same problem with different consequences. The correct application of patterns always requires a careful consideration of the possible alternatives and grounded choice for one of them.

A metaphor will demonstrate this pattern: A good doctor will not immediately subscribe aspirine when one has headache, but will look at all — or the most — possibly relevant symptoms in order to determine the real cause of the headache and the appropriate treatment of it. She will also take possible adverse reactions and interactions with other medicines into account.

Applying the solution of this pattern requires that the students have all needed information available and examine the CONTEXT, PROBLEM AND CONSEQUENCES FIRST. A sensible analysis of the existing problems and the applicable patterns increases the chance of applying the correct pattern. The application of patterns often requires trade-offs [Warren 2005]. The results of such a sensible analysis form the basis for making these trade-offs.

The design patterns described by Gamma et al. [1995] are grouped in three categories: behavioral, structural, and creational patterns. These categories can help with scoping the problem spaces of the specific patterns. So determining the category where a problem actually belongs to could help to decrease the number of pattern candidates which have to be examined. Knowing that there are not that many pattern candidates left can help in the decision to examine them all vs. choosing the first possible solution found.

The problem space should be kept small when applying this pattern in the beginning of a course on design patterns. It is necessary that students first grasp the idea of patterns and do apply them in small scale examples [Pillay 2010].

However, applying BEST FITTING PATTERN CHOICE also includes a discussion of traditional, naïve, and non-patterns-based approaches for solving the problem at hand [Gestwicki and Sun 2008]. This ensures that if a pattern does not offer the best solution or comes for the price of a much higher complexity in the resulting context, then it is better to put SIMPLICITY ABOVE PATTERNS and use one of the traditional or naïve approaches.

One application of this pattern was included in an exercise which was done after introducing the students to the design patterns ADAPTER*,* PROXY*, and* FACADE*. They were then given following problem statement and asked to choose one of the previously taught patterns for solving it: “A client needs only to access the last results and the table of a football administration system". All three patterns were chosen by some students, and we encouraged a discussion about why the different patterns were chosen. It was made clear that probably all of them could be used to implement the required functionality, but that only the* FACADE *pattern matches with the real problem and also communicates that. This way the* *students were made aware of the fact that it is not sufficient to just apply a solution of one pattern, but to make sure that pattern and problem match and therefore the implementation also communicates the intent of the pattern application.*

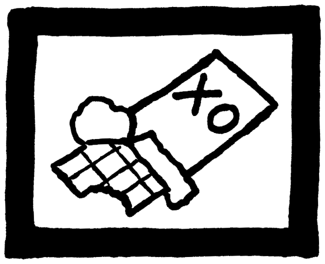
In other discussions the students were constantly asked why they did not use another pattern. The teacher here asked on purpose for incorrect patterns in order to initiate a discussion and create an awareness of making sure that indeed there is a BEST FITTING PATTERN CHOICE*.*

The authors of Head First Design Patterns, Freeman et al. [2004], mention that if before applying a pattern the user should study other patterns first which look like they could be applicable too. Based on the matching of the different parts — they name intent and applicability, which include the problem, context and also consequences of a pattern — with the situation at hand a fitting pattern can be selected.

Bauer and Baumgartner [2012] cite BEST FITTING PATTERN CHOICE *(under the synonym PERFECT FIT) in their forthcoming book on E-Portfolio Patterns. They encourage students to make use of it when applying the patterns from the book for the creation of their E-Portfolio. This is an example from the domain of education.*

Pattern 5: EXPERIENCED ADVANTAGE*

Also known as: Rewarding Sweets.



Let’s face it, a nice creamy chocolate cake does a lot for a lot of people; it does for me.
Audrey Hepburn

You want to show students that patterns are really helpful.

❖ ❖ ❖

It is hard for students to see the advantages generated by correctly applied pattern solutions if they are only told to them. This has negative impact on the motivation of the students to use patterns outside of the educational environment.

After students understood the EXPERIENCED PROBLEM, they start to understand why a solution is needed to solve a problem. But purely applying a pattern is only part of the story.

Shortened Software Lifecycle. The correct appliance of patterns creates a resulting context, which offers in some cases advantages (e.g. better modifiability). Even if the students apply a pattern correct to reach this, they often miss the part that they really get advantage out of this, as the student projects often stop right before getting to this phase of the lifecycle.

Curricula. It seems that computer science and software engineering curricula still focus mainly on the activities prior the deployment and maintenance phases in the lifecycle of software systems.

Belief. Purely hearing from the teacher that a solution is a good solution because it offers some advantages which actually can’t be experienced by the students themselves, requires a high belief of the students in what the teacher says. Some students experienced that some teachers don’t have reasonable arguments for why some things are good and have to be done in the way the teacher wants to. Their belief is therefore dependent on their previous experiences with specific teachers.

Motivation. If students do not believe in the advantages of applying patterns, then they probably get their motivation purely out of the grading or teacher’s feedback they receive. If this this is the only thing of value for them it is likely that they do not apply patterns again or that they still apply them in an incorrect way. This extrinsic motivation is therefore not very helpful in teaching design patterns.

❖ ❖ ❖

Therefore: Give the students rewarding sweets — something of value or satisfaction for the students — by letting them experience the advantages one gets after or during the correct application of a pattern first hand.

In the domain of software design the benefits of applying patterns can — and should — be demonstrated to students. But in other domains this is not always possible, as the real implementation of a pattern solution might be out-scoping the teaching context (e.g. when trying to show the benefit of Alexander's TOWNS pattern). Therefore this pattern is considered as an invariant in the context of software design patterns.

This pattern is closely related to EXPERIENCED PROBLEM, but both patterns cover different aspects. While EXPERIENCED PROBLEM shows what happens when the problem is not addressed, EXPERIENCED ADVANTAGE reveals what happens when the problem is appropriately addressed and solved, using the pattern selected as BEST FITTING PATTERN CHOICE.

EXPERIENCED ADVANTAGE can be realized using an existing implementation of a pattern solution. This implementation can then be used as basis for a follow-up assignment, where the focus is led on making use of the benefits of the correct pattern application. This way the students can experience the advantage themselves. However, the implementation can be of different origins: as PATTERN IMPLEMENTATION MATTERS, the students may already have implemented a pattern in the correct way themselves. But also a solution can be given to the students by the teacher which includes a correct implementation of a pattern.

In some cases it might be sufficient to only discuss the benefits. For example, Gestwicki and Sun [2008] introduced the STATE design pattern by comparing it with earlier designs made by the students and discussing the shortcomings of these designs and the possible benefits from the application of the STATE pattern.

The implementation described in [Cinnéide and Tynan 2004] focusses more on the advantages a pattern can offer during its implementation. They used a problem-based approach in order to let the students "appreciate the flexibility provided by the pattern". The students had to implement a solution without using a specific pattern and then implement it again using this pattern, showing them that using the pattern was the more easy way.

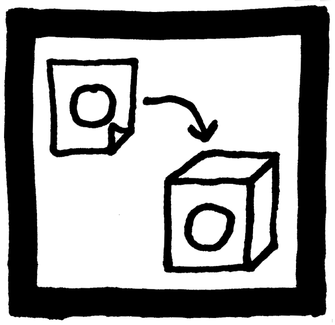
Warren [2005] states that students should experience the full lifecycle of a pattern so that they really appreciate the benefits. Using this pattern reduces the need for projects which cover the full software lifecycle, as it simulates the phases which can not be included in student projects due to time constraints. This increases the awareness of the consequences of applying patterns and also offers a way for introducing activities to the curriculum which are related to phases in the later part of the software development lifecycle.

In one exercise the students were asked to implement a small cook-administration system. All cooks made use of one of a few preparation strategies, and the students were asked to use the STRATEGY *pattern to realize this. In a follow-up assignment they were asked to add another strategy and assign it to one of cooks. Furthermore they had to describe how difficult it was to realize this and how long it took them. Their answers showed that they experienced it as an easy task which was implemented in a few minutes. So they had a noticeable benefit after the application of the* STRATEGY *pattern.*

Cinnéide and Tynan [2004] used this pattern. The students were offered two solutions, one implemented in an ad-hoc way and one using the OBSERVER *design pattern. They were required to add a new view for both solutions. This way they were able to* ***experience*** *the simplicity offered by the solution which used the* OBSERVER *pattern instead of just reading about it, which increases the understanding of the advantages of applying patterns. As stated earlier, this implementation combines* EXPERIENCED PROBLEM *and* EXPERIENCED ADVANTAGE *in one exercise.*

Pattern 6: PATTERN IMPLEMENTATION MATTERS*

Also known as: Implementation Matters.



Knowing is not enough; we must apply. Willing is not enough; we must do.
Johann Wolfgang von Goethe

You want to make sure that students understand how to implement design patterns in a correct way.

❖ ❖ ❖

The students have difficulties with applying design patterns if they only read or hear about them. It is hard for them to add the information necessary for the pattern implementation, which has been abstracted away during the definition of the pattern.

Similar to programming techniques, it is not sufficient to just know about the solution a specific design pattern describes in order to use it in the intended way. E.g. the concept of recursion should be implemented and played with in order to get a real understanding of it. The process of unfolding to specific needs is generally an essential part of using abstractions like patterns.

Abstract vs. Concrete. Design pattern solutions are usually given in an abstract way. The participants are described and the structure is shown in a class diagram. However, all these stay on an abstract level for the student. Even if a sample implementation is given for a specific problem, the students still might have problems on how to apply the pattern for *their own* concrete problem. It is not an easy task applying an abstract solution to a concrete problem.

Implementation Complexity. Some patterns are easy to implement, while others are much more complex. This difference is not easily comprehensible for students.

Code Generation. Some development tools offer support for the application of design patterns[1]. These tools generate most of the code which makes up the important parts of a pattern implementation. The students do often not study this generated code and take for granted that this code reflects the best possible implementation of the pattern solution independent of the rest of their implementation. So if they have to implement a pattern on their own — without using the generating facilities of such an IDE — or if they have to recognize patterns in source code, they are likely to fail.

❖ ❖ ❖

[1]Design pattern generation support can be found e.g. in the Netbeans IDE or in the Eclipse IDE (using PatternBox, http://www.patternbox.com).

Therefore: let the students implement the pattern solution.

With software design patterns it is always possible to implement them, and this helps students to understand the transformation from abstract pattern solution to concrete implementation better. However, this is not always easily possible in other domains. E.g. the patterns of Alexander can mostly not be implemented in the context of a course on architecture, and educational patterns at curriculum level might also be too big to be implemented during a course on didactics. This pattern is therefore considered as an invariant in the domain of teaching software design patterns.

As Ralph Johnson says in [Goldfedder and Rising 1996]: “...people can’t learn patterns without trying them out.". It is hard for non-experienced programmers to realize the power of patterns without a concrete application of them [Astrachan et al. 1998].

The solution is supported by the pedagogical pattern PREFER WRITING [Bergin et al. 2011], which encourages the use of written exercises and includes also the writing of source code.

Having students implementing design patterns on their own helps in getting a better understanding of the solution in general. If a concrete pattern implementation is given to the students, there will be a higher chance that they recognize the participants of the pattern and the roles they play. So even if they use code generation, having implemented a pattern on their own helps them in using it the correct way and to understand the generated code.

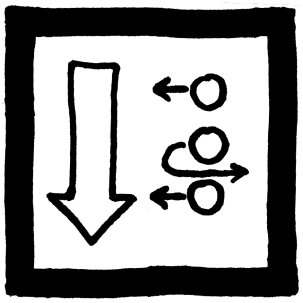
This pattern is just a part of the whole language and should not be used in isolation. PATTERN IMPLEMENTATION MATTERS can be used in combination with EXPERIENCED ADVANTAGE, but does not have to. After using PATTERN IMPLEMENTATION MATTERS it is advisable to also ensure a PRINCIPLE-SUPPORTING PATTERN USAGE and that the students put SIMPLICITY ABOVE PATTERNS.

Probably all courses on design patterns include the implementation of at least a few of them. In our course the students had to implement e.g. the STRATEGY *pattern. This exercise was then also used to give them* EXPERIENCED ADVANTAGE*. Another exercise included the implementation of the* FACADE *pattern. Many more examples can be found in the references section.*

In Head First Design Patterns many implementation exercises are included [Freeman et al. 2004].

Pattern 8: PRINCIPLE-SUPPORTING PATTERN USAGE*

Also known as: Principles Are Leading.



Rules are not necessarily sacred, principles are.
Franklin D. Roosevelt

You want to ensure that students are not applying patterns blindly, but also take the overall design into account.

❖ ❖ ❖

While learning design patterns students often focus on the implementation of the patterns in isolation, which regularly results in a bad overall design.

Good design of an object-oriented system is based on basic principles like high cohesion, loose coupling, etc. They should be taken into account while designing, and design patterns are just a tool to do so. As Warren [2005] states, the focus should be put on *doing* design by making effective use of design patterns.

Small Scale vs. Large Scale. Design patterns are often taught first using small examples. The design principles do not play a prominent role in these examples. However, when the examples become larger also the principles become more important. But the students often apply the patterns as in their small examples, which increases the chance of violating basic design principles.

Golden Design Bullet. Students tend to think that the pure application of patterns does automatically lead to a good design.

❖ ❖ ❖

Therefore: Make sure that the students understand that basic design-principles are more important than the patterns themselves and should therefore always be followed first. Emphasize that design patterns *can support* the implementation of these principles if applied correctly, but do not automatically so.

Not for all domains principles have been identified which form guidelines for creation in these domains. In software design these principles are well known and described, and because patterns are mainly intended to follow these principles, this pattern is considered as invariant in the domain of software design patterns.

This pattern is inspired by the framework process pattern IT'S STILL OO TO ME [Carey and Carlson 2002], which is applicable for the development of frameworks, but also for the usage of design patterns. Rasala [1997] states that one of the problems in teaching design is the “lack of clarity about fundamental issues". This problem is addressed by this pattern as well through making the fundamental issues — the principles which have to be followed — explicit and leading. The correct consideration of the principles should be emphasized continuously by the teacher or trainer when teaching design patterns. This ideally is also reflected when assessing the students' practical work: not the multiple application of different patterns should be rewarded, but the support of the applied patterns for the realization of the design principles in the overall design.

Gestwicki and Sun [2008] define three main learning objectives for teaching Design Patterns. The first and most important one is: “The student understands object-oriented design and modeling". This requires the constant reminding of the students what a good object-oriented design is based on. Through making the OO-principles leading, this learning objective could be more easily achieved.

As stated by Vlissides in [Beck et al. 1996], design patterns are not necessarily object-oriented. So the design principles don't have to be necessarily object-oriented. But the more important part is that whatever paradigm is used, the patterns should be used to support the principles valid for this paradigm.

EXPERIENCED ADVANTAGE can be used for the implementation of this pattern. If shown to students, the EXPERIENCED ADVANTAGE is often of value because of their maintenance of the design principles.

In our course we included in the discussions with the students also the resulting context. The students were constantly reminded of the design principles and had to explain how the resulting context conforms to these principles.

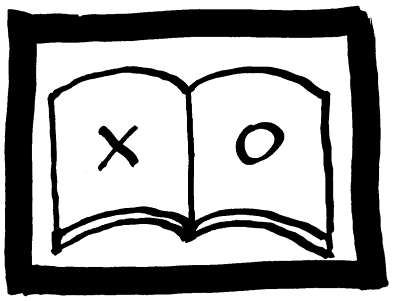
Eduardo Guerra introduces patterns in his courses at the Instituto Tecnológico de Aeronautica as examples of how to realize different principles. This way it is ensured that the focus of the course is primarily on these principles and that the patterns only help in realizing them, which also makes the difference between principles and patterns more clear to the students.

In the book Head First Design Patterns, Freeman et al. [2004] make a distinction between different levels of pattern thinking. They describe the highest level as Zen Thinking, which consists of thinking in object principles and applying patterns only if this naturally follows the object thinking. They also repeatedly emphasize the design principles and explain how the patterns are used to follow them. One example is the principle of loose coupling and how the OBSERVER *pattern can help in realizing this principle.*

Patlets

Pattern 1: PATTERN UNDERSTANDING**

Also known as: Understand Design Patterns.



During the first semesters of their study, students have obtained good knowledge of the basic concepts and techniques of their study. For a computer science student this is knowledge of programming and (object-oriented) principles as well as a good understanding of non-functional requirements like modifiability, reusability, or more general maintainability. You now want to introduce design patterns and make sure that the students understand and apply them as intended.

❖ ❖ ❖

Design patterns are conceptually different from other programming or design techniques, and not taking this into account when teaching them often results in students applying design patterns in an inappropriate way.

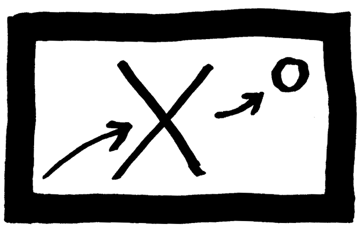
❖ ❖ ❖

Therefore: ensure that students understand *all* aspects of design patterns, their lifecycle, and how their use relates to the overall context by addressing these aspects, the lifecycle and the relations explicitly when teaching design patterns.

This pattern is marked as true invariant, because independent of the domain it is important to understand the whole idea of patterns. Also with patterns of domains other than software design people tend to focus mainly on the solution part. This pattern can be implemented by choosing the appropriate patterns of this language, dependent of the domain and the context the patterns are taught in.

Pattern 2: CONTEXT, PROBLEM AND CONSEQUENCES FIRST**

Also known as: First Things First.



After an initial introduction to patterns, the students will be required to apply them as well. The application requires the choice for a pattern and the application of its solution. You now want to show students how to start applying patterns in a correct way.

❖ ❖ ❖

Students who start to learn patterns often go straight to the solution and apply it, hastily skipping the problem, context, forces, and consequences parts of the pattern.

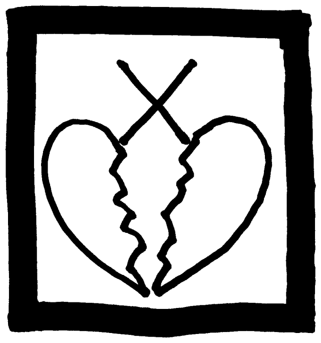
❖ ❖ ❖

Therefore: Focus first on the problem, context, and forces parts of a pattern. Make sure the students understand the need for a good solution. Then introduce the solution and the consequences of applying the pattern.

We consider this pattern a true invariant, as independent of the domain a specific pattern should only be applied after all relevant information has been gathered and analysed.

Pattern 3: EXPERIENCED PROBLEM

Also known as: Feel The Pain.



The students got exercises which require the application of design patterns. Their experience is mainly based on some school projects, and therefore limited. You want to ensure that the students look at the CONTEXT, PROBLEM AND CONSEQUENCES FIRST, but they do not focus enough on context and problem or they do not see the problem as a real problem.

❖ ❖ ❖

Students often apply patterns without understanding why the problem really is a problem. They are not aware of the consequences if this problem is not addressed properly.

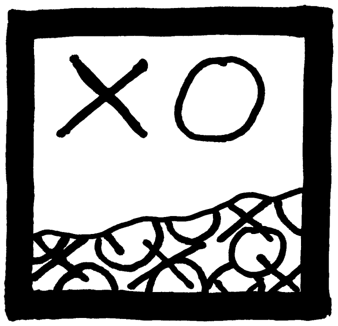
❖ ❖ ❖

Therefore: Let the students experience the problems addressed by a pattern first hand before they implement the pattern. Make sure that they understand what consequences it has to not address these problems.

This pattern is specific for teaching design patterns to students without prior (excessive) design experience. It also can be inappropriate to let pattern users experience the problems addressed by the patterns, e.g. when teaching pedagogical patterns one should not enforce the problems addressed by pedagogical patterns. We therefore do not consider this as sufficiently high-level to be marked with one asterisk.

Pattern 7: SIMPLICITY ABOVE PATTERNS**

Also known as: Keep it Simple.



You want to make sure that the students do not add unnecessary complexity through blindly applying design patterns.

❖ ❖ ❖

While learning design patterns students want to show that they understand design patterns by implementing as many patterns as possible. Most often this adds unnecessary complexity without adding value.

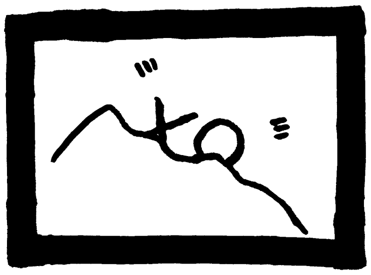
❖ ❖ ❖

Therefore: Motivate the students to always give a rationale for all design patterns they have used. Make sure that they only apply a design pattern when they have not only examined the design problem at hand, but also the consequences of applying the pattern. The application of the pattern should add value to the overall design.

This pattern is considered as true invariant for the teaching of patterns of all domains, as the tendency of applying patterns excessively after being introduced to them can be found in nearly all domains.

Pattern 9: PATTERN DISCOVERY**

Also known as: Discover a Pattern.



The students have enough knowledge of the subject addressed by some of the more simple patterns and they probably already applied the pattern solution without knowing that this was a pattern implementation. You now want to show the students where patterns come from.

❖ ❖ ❖

Students see patterns as something that intelligent people have written. They don't understand that these mostly are captured "best known practices" and that experienced people use them without thinking about them.

❖ ❖ ❖

Therefore: Show students how patterns emerge by letting them discover an existing and well-known pattern by themselves.

We consider this pattern as true invariant, taking into account that it is only applicable for patterns which match with the experience of the students or pattern learners. It was for example used for introducing pedagogical patterns during a lecturer's workshop and also during a presentation at the Dutch Conference on Computer Science Education in 2011 [Köppe 2011b].

Clarification of the Sketches

The main ideas of the sketches are based on the drawings of Takashi Iba which he made on the copy of his print-out of my paper and gave me after the writer's workshop at the PLoP 2011 conference. I drew them myself again and modified some based on my own ideas. The main theme are an X and an O, which mostly represent pattern parts in an abstract way (and with sometimes changing associations). The following list tries to clarify my thoughts for each sketch:

—PATTERN UNDERSTANDING — Teaching patterns is more than just explaining a part of them (mostly the solution). It is therefore not sufficient to study only X or O, but you have to take all relevant parts into account.

—CONTEXT, PROBLEM AND CONSEQUENCES FIRST — Students tend to move on to the solution (the O) quickly without taking the other relevant parts (the usually larger X) into account. So the focus should be on first going to X, and then move on to O.

—EXPERIENCED PROBLEM — The X represents the problem, and the broken heart is used as metaphor for pain.

—BEST FITTING PATTERN CHOICE — The materialized O only fits perfectly into the hole if the diameter, the deepness, the color, and all other relevant aspects of both parts have been taken into account.

—EXPERIENCED ADVANTAGE — After implementing a pattern correctly, considering all the X's and O's, then a sweet reward emerges. The chocolate stands as metaphor for the positive consequences — the advantages — of the pattern application.

—PATTERN IMPLEMENTATION MATTERS — Just presenting the plan of the solution O does not automatically lead to the correct implementation of this solution (in this case the cube with the O). The process of translating this abstract plan to a concrete implementation offers deeper insights in the way pattern solutions have to be implemented in general.

—SIMPLICITY ABOVE PATTERNS — There are many patterns out there, and applying them blindly often leads to complex situations, shown in the chaotic lower part of the sketch. The upper part is more easy to grasp, containing only the things really needed.

—PRINCIPLE-SUPPORTING PATTERN USAGE — Even though it might seem appropriate to apply all the pattern solutions to solve a problem — the O's on the right side — it might happen that some of these solution applications lead to violations of some of the basic principles. These solutions should be initially rejected (as happens with the second O).

—PATTERN DISCOVERY — The metaphor of a mountain and the (unexpected) appearance of a pattern, represented by the X and the O, are used here. Most people know this feeling that when walking through mountains (which is hard to experience in the Netherlands) after reaching the top of a hill, a complete new scenario comes into sight, like e.g. a wonderful sea or a village located in a valley.

Acknowledgements

I would like to thank my PLoP 2011 shepherd Hugo Ferreira. It was great pleasure and honor to receive feedback from the members of the PLoP 2011 pedagogy workshop: Christian Kohls, Takashi Iba, Willem Larsen, Zhen Jiang, Javier Gonzalez-Sanchez, Helen Chavez-Echeagaray, Chikara Ichikawa, Mami Sakamoto, Tomohito Yamazaki, Kana Matsumura, Rinko Arao, Aya Matsumoto, and James Noble.
Special thanks go again to Takashi Iba for giving me the inspiration for the sketches.

REFERENCES

ALEXANDER, C., ISHIKAWA, S., AND SILVERSTEIN, M. 1977. *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series)* Later prin Ed. Oxford University Press.

ASTRACHAN, O., MITCHENER, G., BERRY, G., AND COX, L. 1998. Design patterns: an essential component of CS curricula. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*. ACM, New York, NY, USA, 153–160.

BAUER, R. AND BAUMGARTNER, P. 2012. *Schaufenster des Lernens - Eine Sammlung von Mustern für die Arbeit mit E-Portfolios*. to be published, Waxmann Verlag, Münster, Germany.

BECK, K., CROCKER, R., MESZAROS, G., VLISSIDES, J., COPLIEN, J. O., DOMINICK, L., AND PAULISCH, F. 1996. Industrial experience with design patterns. In *Proceedings of the 18th international conference on Software engineering*. ICSE '96. IEEE Computer Society, Washington, DC, USA, 103–114.

BERGIN, J., ECKSTEIN, J., MANNS, M. L., AND SHARP, H. 2011. Patterns for Active Learning. `http://www.pedagogicalpatterns.org/`.

CAREY, J. AND CARLSON, B. 2002. *Framework process patterns: lessons learned developing application frameworks*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

CINNÉIDE, M. O. AND TYNAN, R. 2004. A problem-based approach to teaching design patterns. *SIGCSE Bull. 36,* 4, 80–82.

FREEMAN, E., ROBSON, E., BATES, B., AND SIERRA, K. 2004. *Head First Design Patterns*. O'Reilly Media.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns*. Addison-Wesley, Boston, MA.

GESTWICKI, P. AND SUN, F.-S. 2008. Teaching Design Patterns Through Computer Game Development. *J. Educ. Resour. Comput. 8,* 1, 2:1—-2:22.

GOLDFEDDER, B. AND RISING, L. 1996. A training experience with patterns. *Commun. ACM 39,* 10, 60–64.

KÖPPE, C. 2011a. A Pattern Language for Teaching Design Patterns (Part 1). In *European conference on pattern languages of programs, EuroPLoP'11, to be published*. Vol. 2011. Irsee, Germany.

KÖPPE, C. 2011b. Een tijd-(en grenze) loze manier van onderwijs: Pedagogical Patterns. In *Proceedings of the NIOC 2011 conference*. Heerlen, Netherlands.

PILLAY, N. 2010. Teaching Design Patterns. In *Proceedings of the SACLA conference*. Pretoria, South Africa.

RASALA, R. 1997. Design issues in computer science education. *SIGCSE Bull. 29,* 4, 4–7.

WARREN, I. 2005. Teaching patterns and software design. In *Proceedings of the 7th Australasian conference on Computing education - Volume 42*. ACE '05. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 39–49.