A Pattern Language for Teaching Design Patterns

Christian Köppe
Hogeschool Utrecht
Institute for Information & Communication Technology
christian.koppe@hu.nl

September 26, 2011

Abstract

Pedagogical Patterns help in general with teaching. But the teaching of design patterns introduces a few special problems like e.g. ensuring that the purpose of patterns is understood and that patterns are applied in the appropriate and correct way. This pattern language for teaching design patterns addresses these problems and offers solutions for teachers and trainers to solve them.

The author would like to have 4 patterns to be workshopped at the PLoP 2011 (the white ones in the language map). However, comments on all patterns are more than welcome, therefore all patterns are also included in this conference version of the paper.

Introduction

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

Teaching is a creational process, in the sense that it creates knowledge, skills, and passion in students. Successful creational processes are based on common patterns, as Christopher Alexander suggests in "The Timeless Way of Building" [1]. These patterns form a language, which can be used to address and solve the problems inherent in this creational process.

Patterns are well known in software engineering, mostly initiated by the publication of the book from the Gang of Four [14]. Many books and papers have been written since, introducing a wide range of patterns and pattern languages and covering diverse fields as design, architecture, requirements, processes, and many others. However, patterns are not always fully understood and applied. Buschmann et al. state that "the very fact that there are many misconceptions, misinterpretations, and mistakes, however, suggests that something is often amiss in the popular perception and definitions of the pattern concept" and that "such misunderstandings inevitably lead to inappropriate application and realization of the patterns themselves" [10]. We believe that one part of this problem lies in the inappropriate teaching of the patterns, which is supported by the experience of other authors [3, 12, 16, 17, 18, 20]. To help in solving this problem we describe in this work a pattern language for teaching design patterns. We see this as an addition to the existing literature, as some aspects — e.g. that patterns should also be taught using large-scale examples — are discussed there in quite detail.

All patterns in this language use the pedagogical pattern Active Student[7] and can also be categorized as pedagogical or — better — teaching patterns. As the goal is to understand design patterns as described by the Gang of Four[14], the students should apply them and experience the full lifecycle of them [23]. To increase the learning effect, the patterns should be taught from different perspectives (using [8]). Different techniques can be used to do this, putting the focus on different parts of the patterns or on different moments in the lifecycle as well as some pattern-specific problems.

This pattern language is aimed at teachers or trainers who want to improve the results of teaching (design) patterns to students or learners in general. Some of the patterns are based on

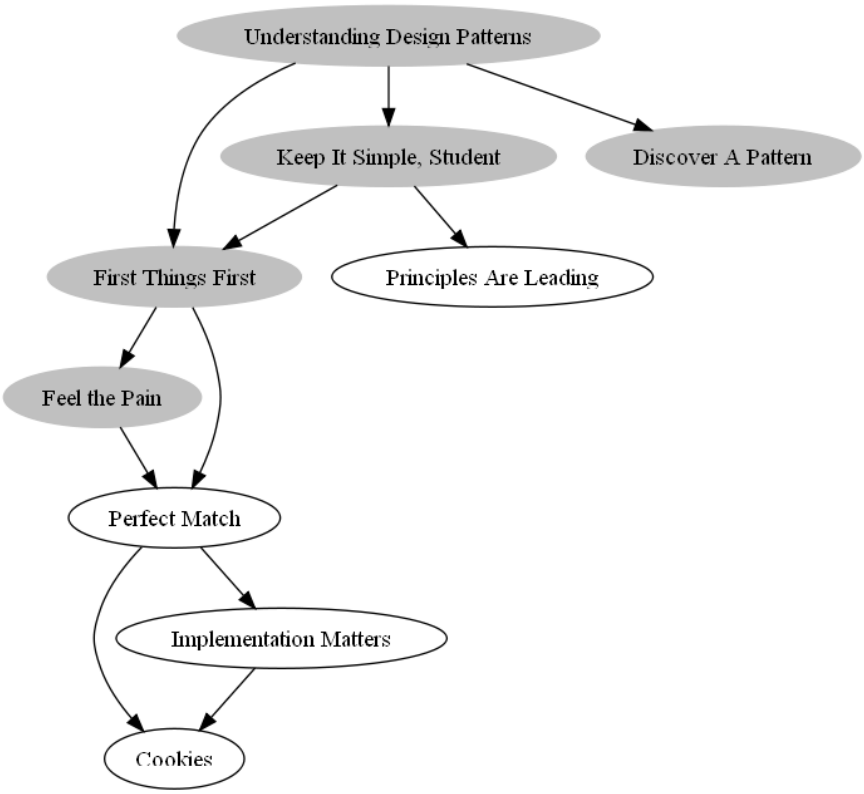

Figure 1: Language Map

experiences made in previously given courses on Design Patterns [19]. Others come from published case studies and experience reports about teaching design patterns [3, 12, 16, 17, 18, 20].

But this language is also useful for pattern learners and pattern authors. Learners can become aware of the obstacles when first introduced to patterns. Authors will find help in some of the patterns — e.g. First Things First or Perfect Match — to also take the learnability of their patterns into account by covering all aspects required for a successful pattern application.

The Pattern Language

Teaching design patterns has a lot in common with general teaching problems. The students need to be actively engaged in order to improve the learning process. Feedback should be given in an appropriate way and different perspectives should be used to enrich the students' experience. These problems are already addressed by different languages published by the Pedagogical Patterns Project [6, 7, 8, 9].

However, there are also some questions specific to this domain:

1. How to make sure that the purpose and the aspects of design patterns are understood and that they are applied in the appropriate and correct way, taking the specific situation into account ?
2. How to guarantee that students still look at the whole design while applying patterns?
3. How to show students that design patterns indeed offer good solutions to their problems?

The answers to these questions are actually corresponding with the learning objectives for the students. The teaching patterns which help in getting the answers are organized in three branches in the same order as the above described questions they answer, where Understanding Design Patterns serves as entry pattern of the language. This is shown in the language map in Figure 1.

The author uses a version of the Alexandrian pattern format, as described in [2]. The first part of each pattern is a short description of the context, followed by three diamonds. 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), 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, shown in italic font, we present some known applications.

The design patterns presented in the GoF-book use a format which differs from formats used in other pattern languages or pattern catalogues [14]. However, the basic parts of patterns are also included in the GoF format. The intent and motivation parts for example describe the context, the problem and some forces and consequences — although to a lesser extent than most other pattern descriptions, as the GoF-patterns are low level and apply only to the relatively small domain of OO-languages. We decided to use the more general terms as introduced by Alexander et al. [2] in this language instead of the GoF terms, namely *context*, *problem*, *forces*, *solution*, and *consequences/resulting context*.

Although some of the patterns in this language could also be applied for teaching patterns of domains other than software design, we decided to not make the language general. Main reasons are some forces specific for the domain of teaching software design patterns, which are directly addressed by the patterns in this language. However, if a pattern could also be used in other domains we will state that in the implementation and known applications sections.

UNDERSTANDING DESIGN PATTERNS

> If you want to make beautiful music, you must play the black and the white notes together.
>
> Richard M. Nixon

During the first semesters of their study, students have obtained good 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 to them and make sure that they understand and apply them as intended.

❖ ❖ ❖

Students often apply design patterns in an inappropriate way. They do not see the consequences of the patterns on the total design and may introduce more problems than they solve.

When beginning with patterns, students tend to apply them blindly without thinking of the overall consequences. It seems to students that some intelligent people invented the design patterns and that using them automatically leads to a good design. Without taking the accessibility of patterns into account, teaching design patterns to students will fail [3].

Abstraction. The concept of a pattern is often not well understood by the students, as patterns are at a higher abstraction level than e.g. programming language constructs or the graphical UML notations. This higher abstraction level makes it harder to understand what a pattern is and how to apply it. But if the overall concept of a pattern — and specifically that of a design pattern — is not understood, then there is a high probability that design patterns are not applied properly.

Incompleteness. Students have to learn a lot of diverging concepts and techniques. These are often loosely coupled and not highly coherent and it is sufficient to have a good understanding of an appropriate subset of them in order to pass examinations or assessments. This is different with design patterns, as applying them incompletely — in the sense of not taking care of all aspects — increases the chance of incorrect application.

Goals. Design patterns are often used to incorporate non-functional requirements of a software system into a design. Understanding the impact design patterns can have on the overall design is

necessary in order to see if the goals are reached and the requirements are indeed implemented. This requires a thorough understanding of the consequences of pattern application.

❖ ❖ ❖

Therefore: ensure that students do understand all aspects of design patterns, their lifecycle, and how their use relates to the overall context.

First of all, the students need to know *all* parts of a pattern. Quite often the knowledge of patterns focuses mainly on the solution. First Things First helps in avoiding this problem. As the students do not have a lot of experience with the problems addressed by the patterns and therefore do not see the advantages the patterns offer, let them Feel The Pain — as result of not addressing the problem properly — themselves. Make sure they find the Perfect Match for resolving their problems and give them Cookies by letting them experience the advantages of a correctly applied pattern. A concrete Implementation Matters hereby, as without implementing a pattern themselves the whole concept of patterns will still stay abstract for students.

The resulting context forms an important part of a design pattern. The problem should be solved without introducing too much complexity, so the students should Keep It Simple. The main goal of design patterns is to help in making a good design in which the Principles Are Leading.

Understanding patterns should include the full lifecycle of them — not only their application, but also their evolution [18, 23]. Patterns emerge through careful observation of good solutions and the extraction of the common parts of these solutions. Applying these technique helps in understanding the patterns, so students should Discover A Pattern themselves.

After the students understood design patterns, their parts and their lifecycle, they can make effective use of them. This will help in improving their designs, but also their design process, as the application of design patterns requires a careful consideration of all aspects of patterns.

This pattern was used as new outline for the course "Patterns and Frameworks" at the Hogeschool Utrecht - University of Applied Sciences. The course was given earlier and some shortcomings were identified. The new structure of the course based on this pattern addresses these shortcomings.
In the beginning the focus is put on the concepts of object orientation and UML. Then a short history of patterns was presented to the students, describing the way Alexander et al. collected their pattern language [2]. The first exercises made use of the patterns First Things First*,* Feel The Pain*,* Perfect Match*,* Cookies*, but also* Discover A Pattern *(see the application sections of these patterns for the concrete implementation). Different exercises also made use of* Implementation Matters*. Later assignments in the course were of bigger scope, so the overall design of the students' solutions was also important. The patterns* Keep It Simple*,* Student *and* Principles Are Leading *were applied in this phase of the course.*

FIRST THINGS FIRST

> Success depends upon previous preparation, and without such preparation there is sure to be failure.
>
> Aristotle

You want to assure that the patterns are applied by the students as intended.

❖ ❖ ❖

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.

Students often think that the obvious way to show that a pattern has been understood is by implementing its solution. This is understandable, as the patterns are often presented with a

strong focus on the structure of the *solution*.

Visual vs. Textual. The structure of the solution of a pattern is often represented with a diagram. Pictures and diagrams are easier to remember than text, so students focus on these while exploring patterns for themselves. Putting the focus mostly on the diagram without examination of the textual parts of the pattern description as well — which contain also the addressed problem(s) and the forces of the pattern — leads to a high chance that they are applying a solution without solving a real problem.

Focus. Many websites which provide information on design patterns[1] give a diagram of the structure of the solution as the first non-text element in a pattern description, which attracts the attention and therefore the focus of the reader as well. So it is not surprising that students tend to look at the solution first and then tend to try to implement this solution without further examination of what their problem consists of. But even experienced software developers often see the diagram as representation of a pattern — the diagram *is* the pattern [10].

Example-based Learning. If the students want to implement a pattern they look for example implementations of it. These example implementations often fall short if it comes to the description of the context and problem this specific implementation addresses, but also what the consequences are after applying the pattern. This encourages the student's perception that design patterns are just implementation techniques.

❖ ❖ ❖

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.

It should become the "natural way" for students to follow the order implied in patterns. They should focus first on the context, the problem, and the forces of a pattern, even if the solution is the first thing which is visually attracting their attention. If examples are used for learning, teach the students to first answer the question of why a pattern is used in this example, and only then to look at how it is applied or implemented. An awareness of the consequences of not respecting this order can be created by letting them Feel The Pain.

To improve the consequences of First Things First, students should actively discuss the problem and context. Gestwicki and Sun state that "a discussion of the domain-specific problem leads to the justification of the design pattern" [16]. Different Active Learning patterns can be used to realize this [7]. However, teachers should be able to facilitate such a discussion. They are responsible for keeping the focus of the discussion on the important parts and preventing the discussion from drifting in an unwanted direction. This requires teachers to have a good knowledge of the design patterns – especially of the addressed problems, the forces, and the consequences as well as possible variations. It is also important to create and maintain an open and constructive atmosphere. Aggressive or attacking behaviour should not be tolerated.

Gestwicki and Sun also emphasize in [16] that the first and most important part of applying a pattern is the good understanding of the context and the problem. The focus should therefore be put on these parts while learning design patterns as well as for the application of design patterns to solve real problems. Wallingford also describes an approach where the first step is to analyse the problem at hand and the context before actually applying the pattern [22].

In one exercise students were required to study the Facade *pattern from the GoF-book [14] and were then asked to summarize this pattern. Most students started to describe the solution, but when asked which problems the pattern addresses the answers became more vague and divergent. So the students got a second exercise (which was the implementation of the* First Things First *pattern). They were given 3 short problem descriptions and had to decide for which of these they would apply the* Facade *pattern. They were encouraged to use the GoF-book and online sources*

[1] E.g. `http://en.wikipedia.org/wiki/Design_pattern_(computer_science)` or `http://www.oodesign.com/`

to substantiate their decision. The students had to work first in groups of two and then in bigger groups of four (an application of Student Design Sprint *[7] in order to improve the communication between the students). Finally, one student of each group was randomly chosen to present the groups argumentation of for which problem description the* Facade *pattern can be applied (and why!) and for which not (and why not!), hereby making use of the pedagogical pattern* Shotgun Seminar *[7]. These argumentations were then subject to discussion of the whole class. This lead to a better awareness of the problem and contexts which were addressed, but also the consequences of applying the pattern. In a follow-up exercise the students then had to implement the pattern. This way all parts of the pattern were covered, and they were covered in the correct order.*

FEEL THE PAIN

> The aim of the wise is not to secure pleasure, but to avoid pain.
>
> Aristotle

You want to ensure that the students do the First Things First, but they do not focus enough on context and problem or they do not see the problem as a real problem.

❖ ❖ ❖

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

The required functionality of a software system can be implemented in countless ways. However, if also non-functional requirements are of importance for the system, it can be assumed that not all of these implementations take these non-functional requirements into account in the same amount. The result will be a system of bad quality which won't be easy to extend, to maintain, to partially reuse, or to adapt. These systems eventually will become a Big Ball Of Mud [13], caused by not addressing the non-functional requirements properly.

Shortened Software Lifecycle. As students' projects are often of limited scope, these non-functional requirements – even if gathered and described in the beginning of the project – are not obviously necessary and of value to students, as the phase where these requirements become important is mostly out of the scope of the project. The need for – and advantage of – the consideration of these requirements during design becomes hence not obvious to the students.

Experience-based Learning. The problems which are addressed by the different design patterns are often ones that students have never experienced by themselves [23]. The consequences of high coupling, low cohesion, low modifiability etc. are not obvious to them, so they often just implement the required functionality without thinking about possible consequences for the non-functional requirements.

❖ ❖ ❖

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.

Experiencing a problem yourself improves the awareness of this particular problem, but also the awareness of why a problem should have been understood properly before solving it. This might require the adaptation of some existing exercises in a way that also the phases of the software lifecycle are covered so that the problems can actually be experienced. In software engineering projects most often this will be the integration or maintenance phase. So possible applications of this pattern could be the task of implementing a new requirement which has huge impact on the current design, or the integration of a new component in a software system.

Astrachan et al. used this technique as well [3]. They based it on the idea that "good design comes from experience, and experience comes from bad design" (attributed to Fred Brooks and Henry Petroski according to [3]).

An important aspect is that the students should not have the experience of failing. Struggling with the problem and having difficulties to solve it are necessary part of the solution, but should not have a negative effect on the students' self-confidence. This requires a sensible application of this pattern: the period where the students actually "feel the pain" should be long enough to make the point, but not longer.

Immediately after feeling the pain of an unsolved problem, the students should look for the PERFECT MATCH and, when found, should be given COOKIES through letting them experience the advantage which the solving of the problem gives them. Astrachan et al. describe this as the *before and after model* [3].

We implemented this pattern in a workshop using Class-Responsibilities-Collaboration (CRC) cards as introduced by Beck and Cunningham [5]. Prior to the introduction of the Observer pattern [14] the students where asked to think of classes and a mechanism which automatically represents changes different values on different display types (which is actually one of the most often found examples of a possible Observer-application). They were required to use the CRC cards for this. After finishing their designs, they had to replay a simple change of one value including the automatic refresh of all representations. Then they had to add another representation and to discuss what they had to change in their design and how difficult this was. This way the students recognized that even a small addition of another display can require a relative big amount of changes.

This pattern can also be found in an exercise Cinnéide and Tynan gave their students [12]. The students had to implement an extension for two different solutions which were given to them. One of these solutions made use of design patterns while the other solution was using an ad-hoc implementation without a good design. The students struggled with the ad-hoc solution, trying the figure out the complex control flow and where to put what code in order to implement the extension. They therefore experienced the maintenance problems generated with this bad design and were able to compare this experience with the one made while extending the well-designed solution, which used design patterns. So this exercise implemented both FEEL THE PAIN *and* COOKIES.

PERFECT MATCH

> 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 solutions.

❖ ❖ ❖

Students often use patterns without exploiting enough 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. So their choice of a pattern is often not appropriate.

One of the underlying problems is that students have difficulties with determining which pattern to use for a specific problem [20].

Solution-focussed. Some design patterns offer similar solutions to 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 [14] – 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.

Applying this solution requires that the students do the FIRST THINGS FIRST. A sensible analysis of the existing problems and all applicable patterns increases the chance of applying the correct pattern. The application of patterns often requires trade-offs [23]. The results of such a sensible analysis form the basis for making these trade-offs.

The design patterns described by the GoF in [14] are grouped in three categories: behavioral, structural, and creational patterns. These categories scope 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 [20].

However, applying PERFECT MATCH also includes a discussion of traditional, naïve, and non-patterns-based approaches for solving the problem at hand [15]. 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 KEEP IT SIMPLE 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 Perfect Match.

COOKIES

> 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 why patterns are helpful.

❖ ❖ ❖

It is hard for students to see the advantages generated by correctly applied patterns if they are only told to them.

After students had to Feel The Pain, 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 belief in the advantages of applying patterns, then they probably get their motivation purely out of the grading or teachers' 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 cookies by letting them experience the benefits one gets after or during the correct application of a pattern first hand.

This pattern is closely related to Feel The Pain, but both patterns cover different aspects. While Feel The Pain shows what happens when the problem is not addressed, Cookies reveals what happens when the problem is appropriately addressed and solved, using the pattern selected as Perfect Match.

Cookies can be realized using an existing implementation of a pattern. 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 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 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 [15].

The implementation described in [12] 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 states that students should experience the full lifecycle of a pattern so that they really appreciate the benefits [23]. 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 used this pattern [12]. 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* FEEL THE PAIN *and* COOKIES *in one exercise.*

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.

❖ ❖ ❖

Design pattern solutions are not learned by only hearing or reading about them or looking at class diagrams.

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.

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[2]. They 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 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.

❖ ❖ ❖

Therefore: let the students implement the pattern solution as well.

As Ralph Johnson says in [17]: "...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 [3].

The solution is also an implementation of Prefer Writing [7], 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. Implementation Matters can be used in combination with Cookies, but does not have to. After using Implementation Matters it is advisable to also ensure that Principles Are Leading and that the students Keep It Simple.

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* Cookies. *Another exercise included the implementation of the* Facade *pattern. Many more examples can be found in the references section.*

KISS (Keep It Simple, Student)

> Any intelligent fool can make things bigger and more complex... It takes a touch of genius - and a lot of courage to move in the opposite direction.
>
> Albert Einstein

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.

The application of a design pattern often adds complexity to the design of a software system in terms of extra classes, methods or relations between objects and classes. But if applied in a correct way, design patterns can add value through improving the quality of a systems' design in respect of non-functional requirements. However, a software designer has to make sensible decisions on when to use a design pattern and how to implement it. These decisions include trade-offs between

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

the added complexity and the improved quality.

Exhaustive Use. Students often seem to skip this trade-off and decide to use a pattern even if it is not necessary. This is probably related to their experiences in other courses, where they are required to show that they understood all concepts and techniques taught to them by applying them. When being taught Enterprise JavaBeans, they probably have to implement all sorts of EJB's. When introduced to specific UML diagrams, they probably are requested to make use of all of them.

Decreased Necessity. With design patterns it is a different story. A typical larger assignment in a course which also teaches design patterns often does require the use of *some* of the learned design patterns, but most probably *not all* of them.

Complex vs. Straightforward. In some cases a problem or requirement could also be implemented — or designed — using a straightforward solution. Most pattern beginners tend to choose for still applying the pattern, thereby adding unnecessary complexity.

❖ ❖ ❖

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.

To ensure that the students always have a rationale let them do First Things First. A good understanding of the problem, the context, and possible consequences form the basis for the trade-offs which are part of the decision on when to use a pattern and when not. It has to be ensured that students understand that *not* applying the solution of a pattern if this is not appropriate is a good practice and higher valued than the simple application of the patterns.

While discussing the trade-offs, experience shows that the students put less emphasis on general principles of a good system design (like low coupling, high cohesion, etc.), so you have to make sure that the students also take into account that Principles Are Leading. Generally, make sure that they understand that the goal is not to apply as many design patterns as possible, but to create a good design that implements the requirements and that design patterns can – and should – help to do this.

We were delivering an earlier version of the course "Patterns and Frameworks", which was not yet based on this pattern language. During the final presentations of the project, one of the students gave as a rationale why he used a specific design pattern that 'they had to use a pattern and that he chose that one'. When asked if he really had a design problem and if he understands the consequences of applying this pattern, he had no answers. After starting to work with this pattern language, the students were required to present their solutions to the whole class and give a rationale for each pattern they were using. The other students were encouraged to ask questions about the design decisions made and also to state if they have possible alternative solutions. This way it became natural to the students to provide a rationale for all their decisions and trade-offs, which lead to decreased complexity of their designs.

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 want to apply them as correct as possible. They hereby often focus on the implementation of the pattern in isolation and forget to look at the overall design and the basic principles.

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 states in [23], 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.

This pattern is inspired by the framework process pattern It's Still OO To Me [11]. It is applicable for the development of frameworks, but also for the usage of design patterns. Rasala states in [21] 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.

Gestwicki and Sun define in [16] 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 [4], 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.

Cookies van be used for the implementation of this pattern. If given to students, the Cookies are 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.

DISCOVER A PATTERN

Nothing is so well learned as that which is discovered.
Socrates

You 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 practices" and that experienced developers use them without thinking about them.

Usually not much focus is put on where patterns come from. They are taught to students (and explained in textbooks) as if they are a mechanism invented by some experts. This does not increase the understanding of what patterns in general are and can do.

Not Invented Here. Some students prefer to design software (in assignments or projects) their own way. They think that patterns are just techniques which are invented by other people, but that they can do better. The students do not know that they already are applying implicitly some patterns.

Complexity Fear. Patterns are at different complexity levels. The Visitor pattern from the GoF-book is certainly more complex and harder to understand than the Singleton pattern [14]. Especially when beginning with patterns, this complexity can be overwhelming. The students are afraid of doing it wrong, so they often decide to not use patterns. This experienced complexity is related to their perception that they have to understand the abstract mechanism of pattern application and that this mechanism is separate from their own programming and design experiences. They do not see both as parts of the same process.

❖ ❖ ❖

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

The experience of discovering something by yourself can lead to a higher learning effect than if it is just explained. While discovering something, one uses his or her already existing knowledge. The parts of the new knowledge can be connected to the existing knowledge, which is the implementation of the pedagogical pattern Expand The Known World. This makes use of Constructivist educational theories where knowledge is built upon existing knowledge [7].

So, if students would discover a pattern by themselves, they can more easy connect and add it to their existing knowledge. They also recognize that discovering patterns is based on the experience of the ones looking for the patterns. James Coplien states in [4] that "it is difficult for designers to appreciate patterns fully, unless they have written one". Warren also describes the experience that one sometimes subconsciously applies patterns while designing [23]. This can be used for letting students make the same experience.

Pattern discovery — or pattern mining — is usually based on many examples. The implementation of this pattern therefore requires the existence of a group of implementations or designs, all solving the same problem. Then the pattern discovery steps as suggested by Alexander in [1, p.254-260] can be used:

> *"In order to discover patterns which are alive we must always start with observation".*

One realisation could be to give students prior to the teaching of a specific design pattern an assignment which includes the problem addressed by this pattern in a well defined context. Let them implement it — without using the pattern! — and present their solution to the class.

> *"Now try to discover some property which is common to all the ones which feel good, and missing from all the ones which don't feel good."*

Discuss all solutions in the class. Let the students vote on which solution or which parts of solutions are the best.

> *"Now try to identify the problem which exists in (...)*[3] *which lack this property. Knowledge of the problem then helps shed light on the invariant which solves the problem."*

Make sure they give arguments for why they've chosen one of the solutions as the best one and how this could be applied to equivalent problems. Then introduce the pattern and compare it with the students' solution. If the example is well chosen and the discussion stays focused on the important parts, the students' solution should be nearly identical to the pattern and, if at all, only differ in

[3] Alexander uses entrances as example. The actual realisation depends on the chosen example.

minor parts. This way, the students discovered the pattern themselves.

This pattern was used in the beginning of a course on 'Patterns and Frameworks'. The students got an exercise where they had to give the design of a solution (with a UML-classdiagram) which solved the problem of maintaining a few cooks with different preparations. It should be easy to give a cook another series of preparations steps. After a short period all groups had to present their solutions and the group was discussing them. Some created a class for every cook (like cookChristian, cookJeroen, etc.), others made one class cook with an attribute name. Some decided to add a list to every cook with all preparation steps, while other solutions included an abstract class preparation with concrete implementations of different preparation strategies. The group decided after a discussion that this last solution was the best one. After that we introduced the Strategy *pattern [14], which only differed in some minor details from the solution which was chosen as the best solution. In some later exercises and assignments the author and another teacher experienced that the students' usage of the* Strategy *pattern was always reasonable.*

Weiss used this pattern as basis for an undergraduate CS course [24]. Using a series of ongoing assignments, all improvements of a small application, he introduced different problems which had to be solved and implemented by the students. The solutions to these problems were subject to group discussions, and the best solutions were then applied by the students. These solutions were actually implementations of specific design patterns, which was communicated to the students after they implemented them. This showed the students that design patterns are indeed good solutions to well identified problems.

Acknowledgements

I want to thank my shepherd Nuno Flores, who helped me to improve the patterns prior to the EuroPLoP 2011. I highly appreciate the feedback which was given during the EuroPLoP 2011 writers workshop members: Christian Kohls, Andreas Rüping, Claudia Iacob, Peter Baumgartner, Reinhard Bauer, and Dirk Schnelle-Walka.
I also would like to thank my PLoP 2011 shepherd Hugo Ferreira.
The comments and questions of Paul Griffioen and Mike van Hilst helped in making this work more accessible, also for non-pattern community members.

References

[1] Christopher Alexander. *The Timeless Way of Building*. Oxford University Press, New York, later prin edition, 1979.

[2] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series)*. Oxford University Press, later prin edition, August 1977.

[3] Owen Astrachan, Garrett Mitchener, Geoffrey Berry, and Landon Cox. Design patterns: an essential component of CS curricula. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 153–160, New York, NY, USA, 1998. ACM.

[4] Kent Beck, Ron Crocker, Gerard Meszaros, John Vlissides, James O Coplien, Lutz Dominick, and Frances Paulisch. Industrial experience with design patterns. In *Proceedings of the 18th international conference on Software engineering*, ICSE '96, pages 103–114, Washington, DC, USA, 1996. IEEE Computer Society.

[5] Kent Beck and Ward Cunningham. A laboratory for teaching object oriented thinking. *ACM SIGPLAN Notices*, 24(10):1–6, October 1989.

[6] Joseph Bergin, Jutta Eckstein, Mary Lynn Manns, and Helen Sharp. Feedback Patterns. `http://www.pedagogicalpatterns.org/`.

[7] Joseph Bergin, Jutta Eckstein, Mary Lynn Manns, and Helen Sharp. Patterns for Active Learning. `http://www.pedagogicalpatterns.org/`.

[8] Joseph Bergin, Jutta Eckstein, Mary Lynn Manns, Helen Sharp, and Marianna Sipos. Teaching from Different Perspectives. `http://www.pedagogicalpatterns.org/`.

[9] Joseph Bergin, Jutta Eckstein, Mary Lynn Manns, and Eugene Wallingford. Patterns for Gaining Different Perspectives. `http://www.pedagogicalpatterns.org/`.

[10] F Buschmann, K Henney, and D C Schmidt. *Pattern-oriented software architecture: On patterns and pattern languages*, volume 5. John Wiley & Sons Inc, 2007.

[11] James Carey and Brent Carlson. *Framework process patterns: lessons learned developing application frameworks*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[12] Mel Ó Cinnéide and Richard Tynan. A problem-based approach to teaching design patterns. *SIGCSE Bull.*, 36(4):80–82, 2004.

[13] Brian Foote and Joseph Yoder. Big Ball of Mud. In *Pattern Languages of Program Design*, pages 653–692. Addison-Wesley, 1997.

[14] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, Boston, MA, 1995.

[15] Paul Gestwicki. Teaching Design Patterns Through Computer Game Development. *Journal on Educational Resources in Computing*, 8(1):1–22, March 2008.

[16] Paul Gestwicki and Fu-Shing Sun. Teaching Design Patterns Through Computer Game Development. *J. Educ. Resour. Comput.*, 8(1):2:1—-2:22, 2008.

[17] Brandon Goldfedder and Linda Rising. A training experience with patterns. *Commun. ACM*, 39(10):60–64, 1996.

[18] Jacqueline Hundley. A review of using design patterns in CS1. In *Proceedings of the 46th Annual Southeast Regional Conference on XX*, ACM-SE 46, pages 30–33, New York, NY, USA, 2008. ACM.

[19] Christian Köppe. Observations on the Observer Pattern. In *Proceedings of the 17th Conference on Pattern Languages of Programs*, PLoP '10, New York, NY, USA, 2010. ACM.

[20] Nelishia Pillay. Teaching Design Patterns. In *Proceedings of the SACLA conference*, Pretoria, South Africa, 2010.

[21] Richard Rasala. Design issues in computer science education. *SIGCSE Bull.*, 29(4):4–7, 1997.

[22] Eugene Wallingford. Toward a first course based on object-oriented patterns. *ACM SIGCSE Bulletin*, 28(1):27–31, March 1996.

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

[24] Stephen Weiss. Teaching design patterns by stealth. *Proceedings of the 36th SIGCSE technical symposium on Computer science education - SIGCSE '05*, page 492, 2005.