A Pattern Language for Teaching Design Patterns (Part 1)

Christian Köppe

Hogeschool Utrecht, Institute for Information & Communication Technology,
Postbus 182, 3500 AD Utrecht, Netherlands
christian.koppe@hu.nl
http://www.hu.nl

Abstract. Pedagogical Patterns help in general with teaching. But the teaching of design patterns introduces a few specific 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.
This part covers 5 patterns of the language in detail — Understand Design Patterns, First Things First, Feel The Pain, Keep It Simple, Student, and Discover A Pattern. The other 4 patterns are covered in part 2 of this work, to be published in the proceedings of the PLoP '11 conference. These patterns are included here as pattlets.

Keywords: Educational Patterns, Design Patterns

1 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 supports the creation of 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 (GoF) [13]. 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. Not much research on the successfulness of the application of these patterns exists, but practitioners often report that patterns are regularly mis- or overused or applied just like other software development techniques with insufficient results. 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]. One part of this problem lies in the inappropriate teaching of the patterns [3, 11, 14–16, 18]. To help in solving this problem we describe a pattern language for teaching design patterns. This language is an addition to the existing literature on teaching design patterns.

All patterns in this language use the pedagogical pattern Active Student [7] and can also be categorized as pedagogical or — better — educational patterns. To increase the learning effect, the patterns should be taught from different perspectives [8] by the teachers or trainers and the students should apply them and experience the full lifecycle of them [21]. 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 experience from previously given courses on Design Patterns [17]. Others come from published case studies and experience reports about teaching design patterns [3, 11, 14–16, 18] and include discussions on when to introduce patterns in a curriculum.

Applying these patterns requires extra preparation time for the teachers or trainers, but students will gain a deeper understanding of design patterns and patterns in general which enables them to succesfully study further patterns themselves. In order to compensate the extra time needed for implementing the patterns of the proposed language one could introduce just a selection of the patterns and leave the other for students' self study.

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 Fit — to also take the learnability and applicability of their patterns into account by covering all aspects required for a successful pattern application.

2 The Pattern Language

Teaching design patterns has a lot in common with general teaching. Students need to be actively engaged in order to improve the learning process. Feedback

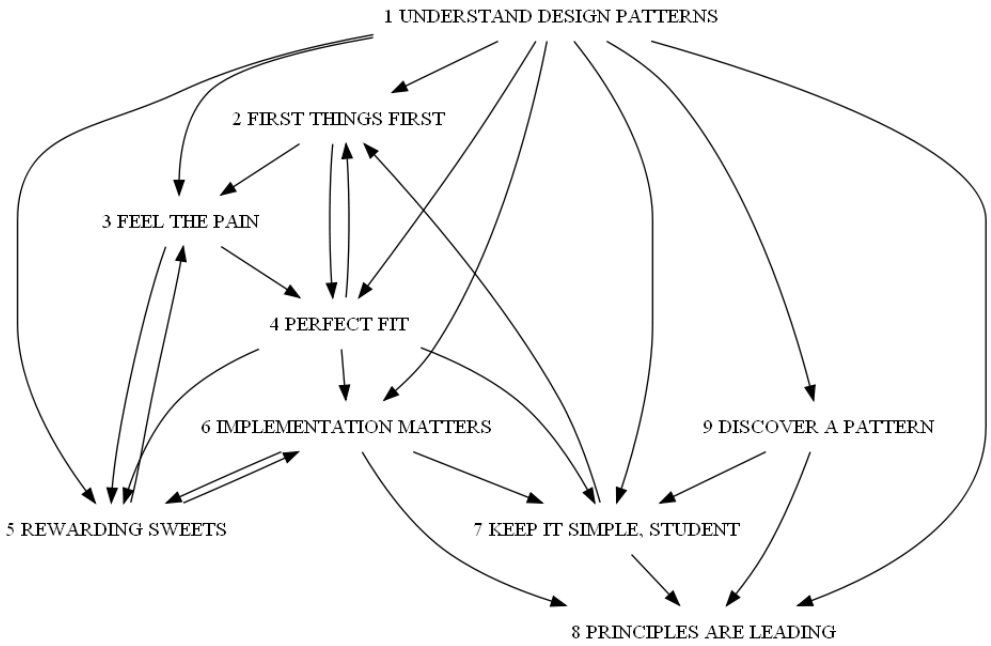


Fig. 1. Language Map

should be given in an appropriate way and different perspectives should be used to enrich the students' experience [6–9].

There are also some questions specific to this domain, which also can be seen as learning objectives for teaching design patterns. The patterns in this language related to the questions are given after each question and are shown in the language map in Figure 1, whereby Understand Design Patterns serves as entry pattern of the language:

1. How can we make sure that the concept of design patterns — their purpose and their different aspects — is understood by the students and that the patterns are applied in the appropriate and correct way by them, taking the specific situation into account? (patterns 2 - 6)
2. How can we encourage students to keep looking at the whole design while applying patterns? (patterns 7 and 8)
3. How to show students that design patterns indeed offer good solutions to their problems? (pattern 9)

This work focuses on the design patterns as described in [13], but is partly also applicable for other patterns.

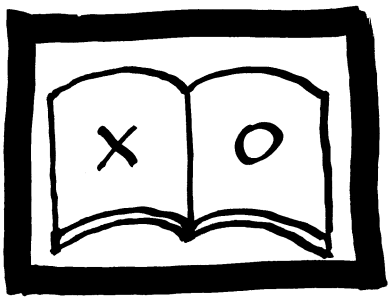
All patterns in this language are just parts of the whole language and should not be used in isolation. Even if the application of separate patterns would add some value relating to the learning objectives, the combination of the patterns will lead to a much deeper understanding in total: the whole is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

These patterns use a version of the Alexandrian pattern format, as described in [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), 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.

The design patterns presented in the book from the Gang of Four use a format which differs from formats used in other pattern languages or pattern catalogues [13]. 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 patterns from [13] 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. in [2] for this language, namely *context*, *problem*, *forces*, *solution*, and *consequences/resulting context*. Using these terms broadens the applicability of the patterns of this language, as all these terms and their related pattern parts can be found in nearly all pattern formats. The described patterns are therefore applicable independent of the format of the design patterns to be taught.

Although some of the patterns in this language could also be applied for teaching patterns of domains other than software design, we did not take these other domains explicitly into account. However, we also successfully applied this language for teaching architectural patterns in a course on Advanced Software Architecture at the Hogeschool Utrecht and some of the patterns (patterns 1, 2, 4, 6, 7, and 9) were also used by the author in a workshop on educational patterns.

Pattern 1: UNDERSTAND 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 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.

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.

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 sometimes 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 realize non-functional requirements of a software system. Understanding the impact design patterns can have on the overall design of such a system 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 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.

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 Fit for resolving their problems and give them Rewarding Sweets by letting them experience the advantages of a correctly applied pattern. A concrete Implementation Matters, as without implementing a pattern themselves the whole concept of patterns will stay abstract for students.

The description of 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 design patterns should include the full lifecycle of these patterns — not only their application, but also their evolution [16, 21]. 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.

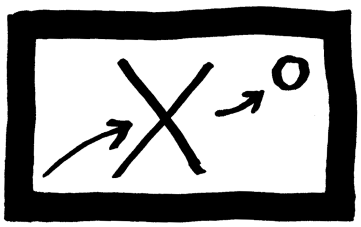
After the students' understanding of design patterns, their parts and their lifecycle has improved, they can apply them more effectively. This will help in

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 Fit, Rewarding Sweets, *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.*

This pattern also forms the main approach used by Eduardo Guerra from the Instituto Tecnológico de Aeronautica when introducing design patterns for the first time.

Pattern 2: FIRST THINGS FIRST



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

You want to show students how to start applying design 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.

Students often think that the obvious way to show that a pattern has been understood is by implementing its solution. This is reasonable considering that 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 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

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

what their problem consists of. But even experienced software developers often see the diagram as representation of a pattern — people think 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 has to become the "natural way" for students to follow the order implied in patterns. They have to 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. This implicitly includes that the needed information is available to the students. 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.

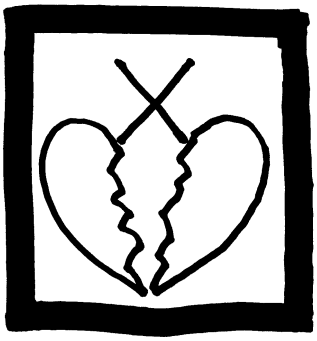
To improve the consequences of First Things First, students should actively discuss the problem and context. Gestwicki and Sun state that "a discussion of the domain-specific problem leads to the justification of the design pattern" [14]. Different Active Learning patterns can be used to realize this [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 unwanted directions. 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. The set-up for such a discussion should also take social aspects into account: it is important to create and maintain an open and constructive atmosphere and aggressive or attacking behaviour should not be tolerated.

Gestwicki and Sun also emphasize in [14] that the first and most important part of applying a pattern is a 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 [20]. Be aware that there are often different levels of problems, ranging from abstract problems like "I need to decouple these two system parts" to concrete problems like "I need to be able to add a new implementation of this algorithm easily". A discussion of the problem part should respect and reflect these levels.

In one exercise students were required to study the Facade *pattern from the GoF-book [13] 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 First Things First*). 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 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 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.*

Pattern 3: 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 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.

The required functionality of a software system can be implemented in countless ways. Only a few of these ways equally respect and balance the non-functional requirements of the system too, while the others often lead to a Big Ball Of Mud [12]. Many problems related to non-functional requirements emerge long after they have been caused.

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.

Sparse Experience. The problems which are addressed by the different design patterns are often ones that students have never experienced by themselves [21].

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 complete failing up to the point of loss of motivation. 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.

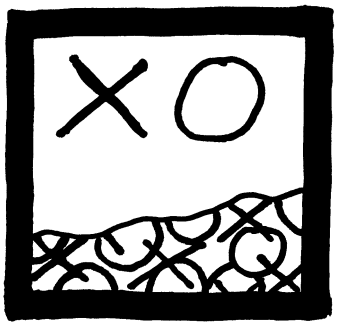
Immediately after feeling the pain of an unsolved problem, the students should look for the Perfect Fit and, when found, should be given Rewarding Sweets through letting them experience the advantage which the solving of the problem gives them. Astrachan et al. describe this as the *before and after 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 [13] 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 large amount of changes.

This pattern can also be found in an exercise Cinnéide and Tynan gave their students [11]. 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* Rewarding Sweets.

Pattern 7: 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 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 (EJB), 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.

❖ ❖ ❖

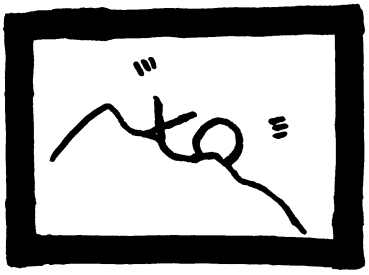
Therefore: Motivate the students to always give a rationale for all design patterns they have used. Make sure that they only apply a design pattern when they have not only examined the design problem at hand, but also the consequences of applying the pattern. The application of the pattern should add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

Pattern 9: 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 [13]. 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 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 and 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 [21]. 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 (...) 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

---

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

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 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 [13], 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 experienced that the students' usage of the* Strategy *pattern was nearly always reasonable.*

Weiss used this pattern as basis for an undergraduate CS course [22]. 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.

Eduardo Guerra lets his students write a pattern themselves based on their prior experience, which is also an implementation of Discover A Pattern.

Pattlets

Pattern 4: PERFECT FIT

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.

❖ ❖ ❖

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.

Pattern 5: REWARDING SWEETS

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.

❖ ❖ ❖

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

Pattern 6: IMPLEMENTATION MATTERS

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.

❖ ❖ ❖

Therefore: let the students implement the pattern solution.

Pattern 8: PRINCIPLES ARE LEADING

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.

❖ ❖ ❖

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.

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 I got from the EuroPLoP 2011 writer's workshop members: Christian Kohls, Andreas Rüping, Claudia Iacob, Peter Baumgartner, Reinhard Bauer, and Dirk Schnelle-Walka.
The comments and questions of Paul Griffioen and Mike van Hilst helped in making this work more accessible, also for non-pattern community members.
Special thanks go again to Takashi Iba for giving me the inspiration for the sketches.

References

1. Christopher Alexander. *The Timeless Way of Building.* Oxford University Press, New York, later print 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 print 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. Mel Ó Cinnéide and Richard Tynan. A problem-based approach to teaching design patterns. *SIGCSE Bull.*, 36(4):80–82, 2004.
12. Brian Foote and Joseph Yoder. Big Ball of Mud. In *Pattern Languages of Program Design*, pages 653–692. Addison-Wesley, 1997.
13. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, Boston, MA, 1995.
14. Paul Gestwicki and Fu-Shing Sun. Teaching Design Patterns Through Computer Game Development. *J. Educ. Resour. Comput.*, 8(1):2:1—-2:22, 2008.
15. Brandon Goldfedder and Linda Rising. A training experience with patterns. *Commun. ACM*, 39(10):60–64, 1996.
16. 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.
17. 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.
18. Nelishia Pillay. Teaching Design Patterns. In *Proceedings of the SACLA conference*, Pretoria, South Africa, 2010.
19. Richard Rasala. Design issues in computer science education. *SIGCSE Bull.*, 29(4):4–7, 1997.
20. Eugene Wallingford. Toward a first course based on object-oriented patterns. *ACM SIGCSE Bulletin*, 28(1):27–31, March 1996.
21. 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.
22. Stephen Weiss. Teaching design patterns by stealth. *Proceedings of the 36th SIGCSE technical symposium on Computer science education - SIGCSE '05*, page 492, 2005.