Using Patterns in Design and Documentation of Software

Th.Gerth, R.Schachtschabel, R.Schönefeld
Technical University of Ilmenau
Department of Computer Science

March 28, 1996

Abstract

In software development, as well as in other areas of life and science, recurring themes and structures are encountered. By expressly describing them together with the underlying considerations and decisions in an abstract manner, i.e. in a pattern form, the knowledge and experience of the developers can be recorded in a structured and systematic way. Such patterns may then be used by other developers to reuse that recorded experience. Furthermore, patterns help to improve the quality of software documentation.
This paper first gives an introduction to the pattern subject. The authors then discuss their experience with using patterns. In this discussion particular attention is paid to some problems associated with pattern employment.

1 Introduction and Motivation

In the last few years the object-oriented paradigm has brought many new essential abstractions and techniques into software development, promising a better control of the software development process. The object-oriented paradigm provides the software developer with concepts, allowing him/her to improve the quality of the software he/she produces. However, these concepts alone do not guarantee that the developed software indeed is distinguished by such positive features as extensibility, reusability and reliability. As in the non object-oriented days, this depends on the ability and experience of the software developers. Using the object-oriented paradigm, they now have to solve the difficult task of building such an object model for the application under development, which would ensure the above mentioned desired features. Still, finding the "right" object models in object-oriented analysis and design is one of the most subjective and creative processes during software development, though at the same time one of the least understood ones. Furthermore, the creative nature of the design phase implies that it is difficult to learn and to teach [Copl95].

The usual and obvious attempt to get around the obstacles of analysis and design is to try to reuse already created and proven designs and architectures. However, the straight reuse, e.g. by using class libraries, is often not possible because of certain system requirements and circumstances. Then, the potentially reusable design has to be modified. This requires that the developer "understands" the designs to be reused, because only then is he able to adapt it to the new problem context.
However, understanding designs is often not easy because the design documentation, if present at all, usually focuses on the "WHAT", i.e. on the result of the design. In most cases the documentation does not include explanations of the underlying considerations and decisions, i.e. the "WHY". However, it is the "WHY", that is essential for the understanding and adaptation of a design [Beck94].

The named problems do not only occur during the initial development of a new system, but also during the maintenance and extension phase of the system. If there are modification requirements, one has to locate those places in the design or architecture, which then have to be changed in order to fulfill the requirements. Again, for that purpose it is important to know why this or that design structure was chosen. Unfortunately, up to now this knowledge is usually only recorded in the brains of the developers, and one must be lucky if the "inventor" of a concerned design structure still is a member of the team.

In fact, in both cases the problem is the same: it is hard to understand and then to adapt a design because unsatisfactory documentations are lacking in descriptions of the design background. The conclusion is to develop and use new forms of design and architecture descriptions which would explain, besides the analysis and design results, the underlying considerations and decisions.

This paper shall point out why and how patterns may contribute to this issues. After an introduction to the relevant terms and definitions, we deal with the potentialities of patterns for software development, the problems arising when working on and with patterns, and possible future evolutions.

The ideas explained in this paper are the result of a comprehensive study of literature in this area, as well as of experiences gained by the authors when using patterns during a research project in co-operation with Allianz Insurance Company, Munich.

2 Terms and Definitions

Patterns are an inherent part of our everyday life and of our environment. Consciously or unconsciously, everyone knows and uses patterns and has a certain notion of what patterns represent and what importance they have for us. This leads to a great variety in the notions and definitions of the term "pattern". Therefore, the purpose of this section is to make clear what we mean when speaking of software patterns.

2.1 Patterns

When looking at the conventional use of patterns we may realize two main **aspects of patterns** [Copl95]:

- By identifying recurring structures, patterns serve as an **analytic means** when recognizing our environment.
  Examples for this aspect are known from such fields of science as biology and architecture.
- In manufacturing products, patterns are used as an **instrument of synthesis**.
  The clothing industry or shipbuilding are typical application areas of this aspect.

Usual pattern definitions in dictionaries like

> "pattern: a model, a template, or a guide for something to be made"

accentuate the second aspect. However, patterns may combine both aspects. And indeed, that is what we are actually interested in for software development: we need patterns that help us to understand software, as well as to build or adapt software.

The following definition meets these needs very well:

> "Each pattern describes a **problem** that occurs over and over again in our **environment** and then describes the **core of the solution** to that problem in such a way that you can use this solution a million times over without ever doing it the same way twice."

This definition is by the architect *Christopher Alexander* [Alex77] who explored patterns in architecture in the 70's, and whose work has influenced many researchers investigating software patterns.

Using the highlighted words we may state this abbreviated version:

"A pattern describes the core of a solution to a problem in a context."

A widely accepted definition among the software patterns community reads:

"A pattern is a solution to a problem in a context."

However, we think that this is an unacceptable abbreviation because it neglects the *abstract nature* of patterns. A pattern does not describe a particular solution. Rather, it describes a solution in a common and abstract way, so that it may be applied to many situations.

In debates about patterns, sometimes there is confusion caused by an insufficient comprehension of the relationship between the "**physical**" pattern, i.e. the recurring structure as found in software, and its description. Some people do not make any distinction between them. Indeed, to some extent it is okay to see them as a unit, because we can consciously reuse a physical pattern which exists regardless of our knowledge, only when it is recognized and described. However, we have found that sometimes it is useful to make a clear distinction between them, i.e. to think of them as different but closely correlated "things". In section 3 we will see when and why.

2.2 Pattern Systems

A single pattern may help us to solve a particular class of problems in system development. However, during this development process we are fighting many kinds of design problems. Therefore, we need many different patterns which may be summed up in a **collection** of patterns for a given problem domain. If this is not a lax but a **structured** collection, i.e, if its patterns are coupled and coordinated, we call it a **pattern system** [1].
The value of a well thought-out and well structured pattern system consists therein, that it may lead a developer through an entire decision-making and problem-solving process. It starts with a problem to be solved in a given context. This problem can be solved by the help of one of the patterns in the system. Thereafter we have a new context with new problems or still unsatisfied constraints. Then another pattern from the system can be applied, whereby, ideally, this pattern is referred to by the previously applied one. At the end of this process, there should be an overall satisfactory solution resolving all the initial problems and constraints [Copl94] [2].

The patterns in a system relate to each other in various ways. For example, one or more patterns may be used to *complete* another one, i.e., each addresses certain problems and constraints unsolved by that pattern. The relationships between the patterns can be used to organize a pattern system, typically by arranging the patterns in several layers. For instance, there could be layers associated with the different phases of the software development process. In this case, the patterns pertaining to analysis would be more coarse than the fine–grained design patterns which are used to complete a solution structure introduced by an analysis pattern.
However, building and organizing a pattern system is not a trivial task. In subsection 3.3 we will discuss some issues related to this topic.

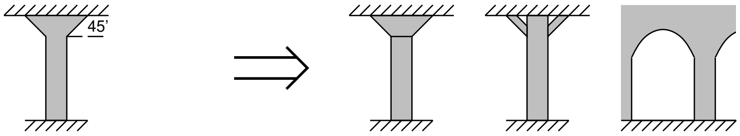
2.3 Examples

Before going further, we will now briefly present a few pattern examples in order to illustrate the above statements regarding the pattern concept (see figure 1) [3].

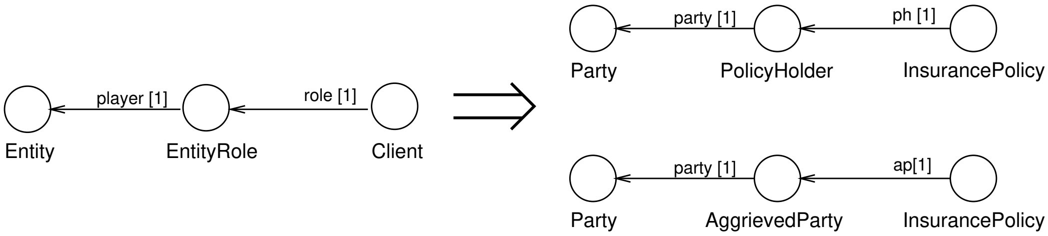
[1] We prefer this term to the other widely-used term **pattern language** introduced by Alexander, who compares the patterns in a system to the words of a human language, from which one may build sentences, paragraphs, and so on.

[2] Because of the capability of a good pattern system to "generate" or to produce a solution (design) for a complex problem within a particular domain, it is often called a **generative** pattern system or language.

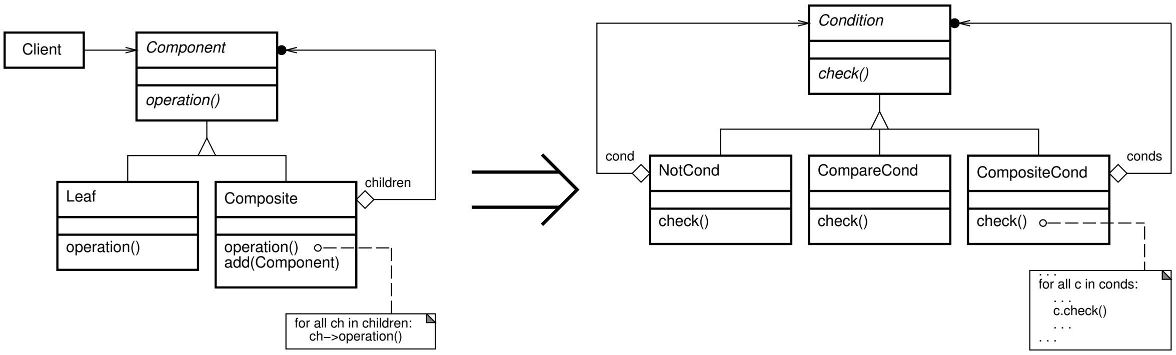
[3] In figure 1, on the left side we show the abstract representations of the patterns, whereas on the right side we present one or more concrete applications of the respective pattern.



a) architecture: column-beam-connection at 45 degree



b) object-oriented software analysis: objectifying roles



c) object-oriented software design: composite objects

Figure 1: pattern examples

To start with a non–software related but well–known pattern, figure 1a depicts an example from Alexander's architectural patterns [Alex77].

Figure 1b presents a pattern which is useful during the analysis phase in system development[4]. It corresponds to a widely applied practice which consists of identifying roles of objects and encapsulating them in separate objects. On the right side we show some uses of the pattern in the insurance business domain.

Last but not least, figure 1c is dedicated to the well–known *Composite* design pattern [Gamma94][5]. The right side shows how it was applied in our recent project in order to model the logical expressions within business rules formulated in a corresponding language.

2.4 The Form of Pattern Description

To round off this tutorial section, some words about the form of pattern descriptions.
The form and the style of a pattern description is very important for the usability of the described

[4] For class diagrams at analysis level we are using the notation by Jacobson [Jacob92].

[5] For design class diagrams we are using a modified and extended Rumbaugh notation [Rumb91] adopted from [Gamma94].

pattern. The description has to be logically structured and comprehensibly but concisely written in order to enable the reader to quickly understand the pattern. The concrete description form may vary from pattern system to pattern system, but it should be consistent within a single pattern system. Furthermore, it should adhere to a widely accepted **basic form**, which might be expressed as follows [Lea95]:

```
IF   you find yourself in CONTEXT
     with PROBLEM,
     for EXAMPLE,
     and are obliged to consider CONSTRAINTS,
THEN for some REASONS,
     apply DESIGN FORM and/or RULE
     to construct SOLUTION
     leading to NEW CONTEXT and OTHER PATTERNS.
```

Although this basic form helps to structure a pattern description, writing a "good" description is not a piece of cake, especially if it must fit with the other pattern descriptions in a pattern system or catalogue. In subsection 3.3 we go more deeply into these issues.

3 Using Patterns in software development

In this section, we show firstly what patterns can do for us in software development. After that, we discuss some issues we have encountered when applying patterns.

3.1 Potentialities

The benefits expected from using patterns in software development may be summarized by the following three points:

- reuse of knowledge and experience

  Experts can use patterns to document their knowledge of "good" (design) techniques they have gained by experience. The pattern form allows them to record that knowledge in a concise and comprehensible way. By reading such a pattern, a non-expert is able to quickly learn from the experts. Based on an entire pattern system for a given domain, laymen are in a position to develop "good" software systems.

- quality improvement of software documentation

  Patterns can help to resolve the contradictory demands for *compactness* and *completeness* of software documentation. The latter one requires, following the introductory section, that the considerations and decisions which have led to the final design of the documented software have to be recorded. If the design process involves the use of patterns, and if a single pattern is applied repeatedly in the documentation it is sufficient to refer at the appropriate places to the corresponding pattern description, which explains the underlying decisions. This way, explaining a recurring decision process each time it is applied in the design can be avoided. Thus, the documentation is complete and compact.

- facilitation of the communication between developers

  If there is a set of accepted patterns among a group of developers (team, company ...), the patterns serve as a common vocabulary, which eases the communication between the developers. Ideas of a certain design structure do not have to be circumstantially explained because it suffices to name the pattern in mind.

Something more could be said about the potential for using patterns in software development. Furthermore, it would be nice to explain how the pattern notion relates to other notions and concepts relevant to software development, such as frameworks, methodologies, creativity, rules, etc. However, there is enough said about these topics in the pattern literature (e.g. [Copl94], [Lea95], [John94]). Therefore, now we are directly going to some problems which arise when using patterns and which require further research in order to make patterns more useful for software development.

3.2 Coping with Thousands of Patterns

As more and more work and investigation is done in the pattern field, more and more patterns are recognized and described. An impressive proof of this evolution are the PLoP conferences in 1994 and 1995. Whereas in 1994 there were about 10 contributions, one year later already four times as many papers were collected. This expresses the huge interest in patterns. However, this evolution has its risks too. If we do not find appropriate ways to manage the increasing number of pattern descriptions, it will be possible that they may confuse the pattern users more than they help them to solve their problems.

The observation, that for an intrinsic single pattern there exists several descriptions (potentially with different names), might lead to the suggestion to minimize the number of pattern descriptions by agreeing on a set of **standard** descriptions.
However, we think that it is nearly impossible to find the "one and best" description for a pattern. Furthermore, when seeing patterns not as single items but in connection with pattern systems, it is neither desirable nor required to tie oneself down to one description. Whereas one description is more suited to the context of one pattern system, another description might fit better with another system. Sure, there are some basic design patterns (see [Gamma94]) as well as some patterns devoted to software architecture (e.g. MVC) which are widely used in many domains of software applications. Regarding these patterns, it seems appropriate to agree on some **reference** descriptions. Then, these might be referred to by descriptions actually explaining the same (physical) pattern but suited to a particular pattern system.

In our opinion, a better way to help the user through the world of patterns, other than restricting the number of "allowed" pattern descriptions, is to present him the patterns within pattern systems. Were there pattern systems for many domains, the user could pick up the system corresponding to the domain of his interest. So, the developer would be confronted only with a restricted set of patterns sufficient for his task. However, building a good pattern system is not trivial and the next subsection is dedicated to this topic.

3.3 Building and Organizing Pattern Systems

It has already been stated that good pattern systems would be of a great value in software development but that it is a hard job to write such systems. This is illustrated by the fact that in pattern literature there are many contributions addressing single patterns or pattern collections but just a few concerned with pattern systems. And these few pattern systems only address rather small problem domains like checking user input [Cun94]. At present, we do not know of a successfully used pattern system for a larger business domain, such as the insurance business. This means, from software development practice still there is no evidence for the claim that a pattern system may lead a developer through an entire decision making and problem solving process ending with a complete system design (see subsection 2.2).

But why is it so hard to compose a good pattern system for a considerable domain? We see three main reasons:

- The patterns in the system have to be carefully described in order to complement and build on each other.

- The complexity of the domain makes it nearly impossible for one person or a small team to cover all the problems and solutions in the domain.
- The variety of relationships between the patterns makes the organisation of the system more difficult.

Regarding the first two points, there is not much else we can do other than gather together talented people who not only know their field thoroughly but who are also able to explain their insights in a logical and comprehensible manner.
To get things started, we propose to line up such work groups within bigger companies and organizations. This follows from several considerations. First and most important, finding patterns in a domain requires an adequate amount of existing software designs to be observed. However, companies tend to hide designs from the public in order to protect their investments. So it is natural to line up pattern work groups within a company where all designs are revealed. Secondly, a company usually works in a restricted number of business domains. This helps to narrow the domain of the pattern system, in accordance with the previous subsection. Thirdly, bigger organizations have the financial power to make the required research investments. And last, they have enough software developer staff, who can evaluate the practicability of the pattern system under work by using it in their daily work.
After the pattern system has reached a certain degree of maturity, it may serve as a kind of developer's guide within the organization. Of course, the pattern system requires perpetual improvement because, as time goes on, previously unknown problems will arise in the domain and new and/or better solutions to known problems will be found.
Time will show whether the companies will be willing to publish their pattern systems in order to get into scientific exchange with other organizations working in the same domain. There is some hope for that in view of the evolution in OMG's Business Object Management Task Force. Building business domain-oriented pattern systems may support the work of this group in finding and defining standard business objects communicating with each other beyond the boundaries of a single organization [Roesch95].

Now to the point of the **variety of relationships** between patterns, which makes the organization of a pattern system difficult. Some well–known relationship types are [Zim94]:

- Patterns may be *combined*.
- A pattern may *use* another within its solution.
- A pattern may be *similar* to another in its problem and/or solution.
- One ore more patterns may *complete* another in order to resolve unsatisfied constraints and subproblems.

These various types of relationships lead to a multi–dimensional structure of the pattern space which is difficult to map to the one–dimensional structure of literarily presented pattern systems. When building a rudimentary pattern system during our above mentioned project, we followed Alexander's approach to use the completion relationship as the main guide in ordering the patterns in the system, and it worked quite well. This is not astonishing if the purpose of pattern systems – to be a guide to system development in a domain – is called to mind. Typically, this development starts with a more or less big problem within the domain and certain constraints. First, the developer needs to find an overall, large-scale solution. Then he works down to the details of design. Therefore, a pattern system is best organized in layers, whereby the higher ones offer large-scale solutions which may be completed by patterns from the underlying layers. To be concrete, at present we have the following four layers in our pattern system:

- Analysis

  Patterns in this layer provide solutions to problems which we assign to the analysis phase. An example is the pattern "Management of Entities" which describes a solution to the typical use case, where a user searches for, views, modifies, or puts in data of business objects – "entities" in the terminology of Jacobson [Jacob92] – for instance "customers".

- Design-I and -II

  Accordingly, these layers contain patterns addressing problems associated with design. Thereby, patterns in layer Design-I describe more complex solutions, e.g. "Transparent Storing of Objects". Design-I patterns can be used, together with other patterns, to complete, for instance, the above mentioned analysis pattern "Management of Entities". Often, the solution offered by a pattern in layer Design-I applies one or more patterns from layer Design-II describing more basic solutions such as the pattern in [Gamma94]. For instance, the solution of Design-I pattern "Platform Transparent Interface Objects" contains a combination of Design-II patterns "Bridge" and "Abstract Factory". However, more important is that Design-II patterns may be applied to complete patterns in the overlying layers, i.e. Analysis and Design-I. For instance, there are several Design-II patterns ("Abstract Factory", "Prototype"), which can be used to refine the *InterfaceKit* component of Design-I pattern "Platform Transparent Interface Objects".

- Implementation

  We introduced this layer for patterns presenting platform or language specific realizations (completions) of patterns in the overlying layers. For instance, in the pattern "Meta Visitor" we describe an elegant realization of Design-II pattern "Visitor" in an environment providing meta information.

Consequentially, the patterns in the system are ordered along the completion path, i.e. beginning with the largest and most complex ones (Analysis), down to the smallest and simplest (Implementation). This ordering is retained not only between the mentioned layers but also within these. For instance, in layer Design-I, pattern "Abstract Table Manager" follows the above mentioned pattern "Transparent Storing of Objects" because it is used to complete the latter one.

Other topics, worthy to discuss in this context, but which we must drop for reasons of time and space, include:

- If and how to describe rules and principles as patterns?
- How to describe alternative solutions?

After all, with modern hypermedia systems it has become much more convenient to cope with the complexity of pattern systems. So, a user may easily jump from pattern to pattern along links associated with relationships of arbitrary type. However, it's still important to establish and document a "logical" order in the system to help the user to orientate himself within the system and to find appropriate starting points.

3.4 Using Patterns for Design Documentation?

In our project we used the pattern format not only to document proven and applied patterns but also to record new designs and their underlying decisions. This practice is not commonly accepted in the pattern community because, by definition, patterns describe the core of solutions repeatedly proven to be successful. Or in other words, patterns may not be invented, but only observed from existing software.

However, sometimes there are situations where it seems that a particular design, developed during software construction, is "good" not only in the first place, but might be reused – with variations – as well in other similar places. We think it is then legitimate to use the pattern format to capture the design and the accompanying decisions in an abstract manner. We regard such solutions as *potential* patterns. Future development has to show if they really are. If needed, they may be rewritten, merged into better ones, or may even be dropped.

To support this approach, following Alexander we mark the patterns in our systems with different degrees of maturity. This gives the reader a notion of how strong we are convinced of a particular pattern.

4 Outlook and Closing Remarks

In general, the field of the use of patterns in software development is a quite young one. As demonstrated in this paper, we are persuaded of the usefulness of patterns. However, software development practice still has to provide the evidence, and still much research has to be done to find answers to open questions associated with the use of patterns, especially of pattern systems. Some of these questions were addressed in this paper, others not.

Finally, we want to call your attention to the area of **expert systems** where we think pattern systems might be very useful, and where in the future they might find a large area of application. As you may have noticed, much of what was said in the paper was not restricted to software development. We have done so deliberately, because in our opinion patterns and pattern systems may be applied to many domains, not only software development. We have already referred to the work of Alexander in the field of architectural patterns.
If we regard a pattern system as the snapshot of the knowledge and experience in a particular domain, and if we accept the potential of the pattern format and of pattern systems to allow a user to quickly learn the concepts and find solutions to a given problem, then it is obvious to use pattern systems as a key component in expert and/or teaching systems. Using the keywords, typed in by the user and describing the problem to be solved, the system might offer one or more patterns possibly providing a solution to the problem. The user can then view and evaluate these patterns. Additionally, by using the hyper-media links of the system, he can explore further patterns which are similar or which may be used to complete the pattern at hand.

References

[Alex77] Chr.Alexander: *A Pattern Language*. Oxford University Press, New York, 1977

[Beck94] Kent Beck: *Patterns Generate Architectures*. Proceedings of ECOOP'94

[Cun94] Ward Cunningham: *The CHECKS Pattern Language of Information Integrity*. Proceedings of PLoP'94

[Copl94] J.O.Coplien: *Software Design Patterns: Common Questions and Answers*. AT&T Bell Laboratories Technical Memorandum, 1994

[Copl95] J.O.Coplien: *Software-Entwurfsmuster*. OBJEKTspectrum Nr.6, Januar/Februar 1995

[Gamma94] E.Gamma et al: *Design Patterns: Elements of Reusable Object–Oriented Software*. Addison–Wesley, 1994

[Jacob92] I.Jacobson et al: *Object–Oriented Software Engineering: A Use Case Driven Approach*. Addison–Wesley, 1992

[John94] Ralph E.Johnson: *A Report on PLoP'94*. Reports on Object-Analysis & Design (ROAD), No.3

[Lea95] Doug Lea: *Patterns-Discussion Frequently Asked Questions*. WWW page, Update March 1995

[Roesch95] M.Rösch: *Business–Objekte vereinfachen die Struktur von Informationssystemen*. Objekt Spektrum, Juli/August 1995

[Rumb91] J.Rumbaugh et al: *Object–Oriented Modelling And Design*. Prentice–Hall, New Jersey, 1991

[Zim94] W.Zimmer: *Relationships between Design Patterns*. Proceedings of PLoP'95