Conceptual Content Management for Pattern-based Software Design: An E-Learning Experience

Hans-Werner Sehring, Sebastian Bossung, Patrick Hupe, Michael Skusa, and Joachim W. Schmidt

{hw.sehring,sebastian.bossung,pa.hupe,skusa,j.w.schmidt}@tuhh.de
Software Systems Institute (STS)
Hamburg University of Science and Technology (TUHH)

Abstract. Modern software engineering masters its complexity problems by applying well-understood development principles. It was the adaptation of design patterns which caused a significant improvement of software design and is one remedy of what was formerly called the software crises. Due to their regular structure and orthogonal applicability the application of design patterns can serve as one class of use cases for software development tools. Design patterns and their utilization constitute an increasing body of knowledge in software engineering. Their regular structure and the availability of meaningful examples make design patterns well-suited for organizational memory and e-learning environments. Patterns are defined and described on two levels [7]:

- by real-world examples, i.e., content on their principles, best practices, structure diagrams, code etc.
- by conceptual models on problem, solution, consequences etc.

This intrinsically dualistic nature of patterns makes them good candidates for conceptual content management (CCM). In this paper we report on the use of the CCM approach for the realization of a CCM system for teaching and training in pattern-based software design as well as for the support of the corresponding e-learning processes.

1 Introduction and Motivation

The entire field of modern software engineering (SE) is a diverse and complex endeavor. Many advances had to be made to master what we use to call the software crisis. One important key to success was finally found in software patterns which introduced another level of abstraction in software design processes and decreased the complexity of designs considerably. Since their first publication in the early 90ies [7] design patterns (and the idea of software patterns in general) have been quickly adapted and are today in wide-spread use.

As modern software systems become ever larger and more complex most software development paradigms agree that some planning-ahead is needed in order to successfully carry out a software development project [22]. While requirements for the application under development are collected during an initial

application analysis phase the subsequent design phase aims at a coarse design of the system under development. In this light design patterns are abstractions over pieces of software with shared design requirements thus helping to exploit accumulated design experience.

Design patterns turn out to be essential pieces of knowledge for software engineers and, consequently, have to be covered by software engineering curricula. As the mere awareness of a design pattern does not enable a software engineering student to appropriately apply it, the pragmatics of pattern application should also be taught. This can be achieved by providing pattern definitions together with best-practice pattern applications and by enabling students to relate, publicize and discuss their personal pattern applications in such definitional contexts. Students can thus actively participate in the design process resulting in an improved learning experience.

In previous projects we have used our Conceptual Content Management (CCM) approach to provide extensional concept definitions and to embed such CCM material into learning environments—albeit in political iconography [20]or art history applications [2]. In this paper we introduce a conceptual model for design patterns and show how previous experiences with active learning can be carried over from neighboring fields to SE. We discuss how existing work on patterns (e.g., [7]) can be represented, communicated and applied by CCM.

The remainder of this paper is organized as follows: We commence with an overview over requirements to SE tools and their relationships with content management in section 2. In section 3 we provide a conceptual model suitable for describing patterns in learning systems and also report on some related work. Section 4 describes pattern learning scenarios with particular emphasis on active learning in conceptual content management systems. We conclude with a summary and outlook in section 5.

2 Requirements for Software Engineering Tools

As a basis for discussion this section contains some remarks on SE processes, in particular on the pattern-based design of software, and we argue that SE—especially the aspect of conserving design knowledge—shares properties of advanced content management tasks.

2.1 Software Engineering Processes

A typical SE process consists of several phases: analysis, design, implementation and test (e.g., [22]). During these phases, numerous artifacts are created. These artifacts stem from different models and coexist for various reasons: different levels of abstraction, perspectives, revisions, or alternate representations.

Typically, artifacts of later phases are created from higher-level artifacts of the previous phases, but dependencies are not linear and difficult to trace: Diverse analysis requirements, platform constraints and modeling acts such as architectural decisions, pattern applications, etc. manifest themselves in later

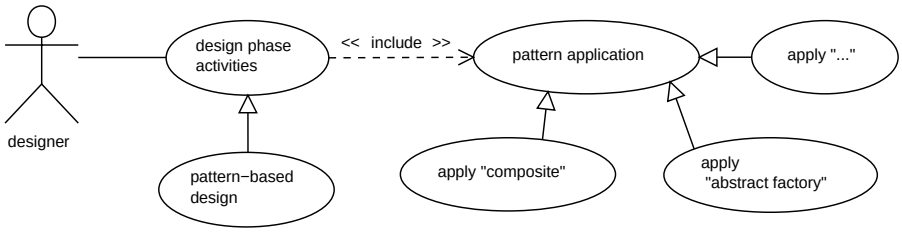

Fig. 1. Design use cases

artifacts. While the knowledge about these manifestations is available at the point of decision-making, this information often is not part of a SE process [10]. Instead, such information has to be recorded in additional documentation which accompanies the process. Only recently efforts have begun to interconnect these artifacts at least partially [16]. The conservation and communication of development experience is an issue not systematically addressed by SE.

2.2 Design Phase Activities

One of the phases of typical SE processes is that of design. This phase gains lots of attention because it is the point where reuse of development experience can take place at a high level.

The application of design patterns has become an important member of the set of design activities (fig. 1). Reasons are that such applications are well-understood. Design patterns constitute a growing body of knowledge and best-practices. They can be used to preserve and communicate design experience.

These reasons—design patterns being well-understood and being a medium to communicate development experience—makes them a germane means for teaching software development. We apply patterns in SE education as a first object of study towards a model-based treatment of design patterns. As a first step in this direction we treat specific patterns (see fig. 1).

2.3 Software Engineering as a Content Management Activity

We have studied how entities are represented by means of content management [18] also with particular regard to SE [3]. Emphasis is put on the representation of entities that cannot fully be represented by data records alone, especially items that have subjective interpretations. We refer to the research of such content management applications as *conceptual content management* (CCM for short).

Based on the epistemic observation that neither content nor concept exist in isolation, CCM is based on the conjoint representation of entities by pairs of content and conceptual descriptions. For pairs of these two we introduce the

notion of *assets*. *Content* is presented by arbitrary multimedia documents. *Concepts*—following semiotics—consist of the *characteristic* properties of entities, *relationships* with other assets which describe the interdependencies of entities, and *constraints* on those characteristics and relationships.

The statements of the asset language allow the definition of asset classes, the creation and manipulation of asset instances, and their retrieval. Some examples follow. For a more complete description of the asset language see [21].

The following sketches asset definitions for a SE processes:

```
model GeneralSoftwareEngineering
class SoftwareModel {
  content xmiDocument :org.w3c.dom.Document
  concept relationship classes   :ClassDescription*
          relationship sequences :Sequence*
          constraint operationsDefined
            sequences.objActs.msgs <= classes.operations.name
            and ... ; matching signatures
          ⋮                                                    }
class Sequence {
  content topNode :org.w3c.dom.Element
  concept relationship objActs :ObjectActivation* }
class ObjectActivation ...
```

In the example, an XML document is the content of a general `SoftwareModel`, presumably an XMI representation of UML diagrams. Related instances of further asset classes are used to describe parts of the software model—classes, sequences, etc.—in more detail.

SE entities are described by the aforementioned characteristic attributes and relationships. Possible types of content handles and characteristics are determined by an embedded language which also is the target language of the model compiler (currently Java). More extensive use of the asset model is achieved by employing constraints that reflect the rules of the chosen SE process. In the example this is demonstrated by a constraint `operationsDefined` which in addition to the semantics of the UML ensures that for each message sent according to a sequence diagram a method with a matching signature is defined.

Since assets are especially designed to support subjective views, asset classes as well as instances can be *personalized* by means of redefinitions on an individual basis. Based on the above definition of `SoftwareModel` a user can define

```
model MySoftwareEngineering
from GeneralSoftwareEngineering import SoftwareModel
class SoftwareModel {
  concept relationship objects :ObjectDescription*
          constraint objects.type <= classes       }
```

to explicitly refer to instances and to require that all of their classes have to be part of the model. Anything that is not mentioned in the personalization remains the same as in the original definition.

For asset redefinitions there is a demand for *openness* and *dynamics*. We call a CCM system (CCMS) open if it allows users to define assets according to their current information needs. Dynamics is the ability of a system to follow redefinitions of assets at runtime without interrupting the users' work.

To account for dynamics, our approach to CCM consists of three main contributions [17]: an *asset language* for the description of entities by both content and conceptual expressions, a *modularized architecture* for evolving conceptual content management systems, and a *model compiler* which translates expressions given in the asset language into CCMSs.

3 Modeling Design Patterns

3.1 Design Patterns

The central idea of design patterns is to capture solutions to recurring problems. In other words, experiences gained by adept programmers are put into a form that is suitable to pass on these experiences to others, thereby eliminate the need for them to relearn the same knowledge "the hard way". The exact workings of this mechanism are currently under research, see [10]. This capturing of shortcuts in learning experiences makes design patterns an important part of SE curricula.

Existing work on design patterns (most prominently Gamma et al, [7]) identifies four central elements of a pattern: A name, a context in which it can be applied, the solution it provides, and the consequences that arise from it. However, [7] also acknowledge that there is some subjectivity in design patterns. The authors not that what is and what is not a pattern depends on the individual user's point of view. Other work [24] also points out the common definition of a design pattern being "a solution to a problem in a context" should be extended to also include: information about Recurrence as well as about Teaching, to provide the means to apply the solution to new situations and to notice that an opportunity to do so has arisen in the first place. This is particularly important with respect to teaching patterns, where a definition of the design patterns is not sufficient. We will describe in section 4.3 how these aspects are handled by our CCMSs.

3.2 Pattern Description Languages

The need for a pattern description language arises in several context which can broadly be divided into those that aim to support the task of creating software (e.g., generatively) and those that inspect existing software (e.g., to discover existing patterns).

A first metamodel for pattern-based CASE tools in proposed in [15]. Based on this metamodel, pattern instantiation is proposed, relating available patterns to actual application artifacts. UML also offers the collaboration mechanism which can to some extend be used to model design patterns. By itself this is too weak

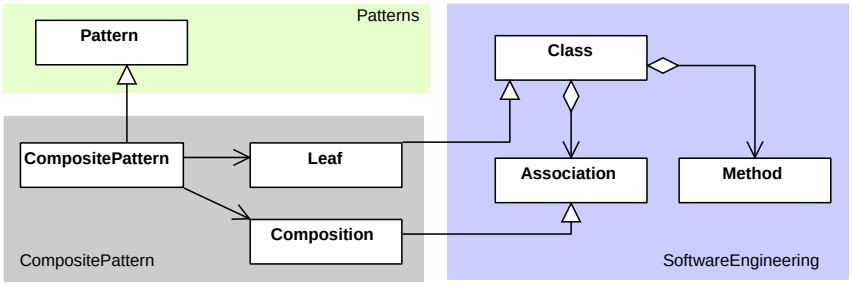

Fig. 2. Classes from different models are combined to model patterns. Most of the model is omitted in this figure for conciseness.

a model for pattern-based tools, which need additional means, e.g., OCL [23]. Yet other tools let programmers work on different levels of abstraction allowing them to work on the source code as well as to instantiate patterns [6].

Pattern extraction from existing software is also of interest (e.g., [8]). The aim is to identify micro-architecture, part of which are patterns. To this end an XML repository is created that can store such micro-architectures by modeling classes in roles. The MVCASE tool for design pattern application [12] uses XMI to describe patterns. The description is geared towards system supported application of the pattern. Taking this one step further, there are approaches to enforce the use of design patterns, e.g., [11].

These approaches offer metamodels for design patterns with several different foci, but there is a large overlap at their cores. Commonly no clear distinction between pattern description and the general object-oriented metamodel is made. Our metamodel for patterns is loosely similar to most existing models, but aims to improve the separation of general pattern description, object-oriented metamodel and specific pattern descriptions for learning.

3.3 A CCM Model for Pattern-based Design

Design patterns are generally presented in a semi-structured manner. Gamma et al identify four essential parts of a pattern: name, problem, solution and consequences [7]. Further substructuring of these elements is not prescribed, even though most authors try to adopt a uniform heading structure. However, there are also approaches that fully formalize the description of design patterns such as [14]. This can provide well-defined semantics for the descriptions as well as reasoning on them. Such formalizations are hardly useful in teaching patterns as learners need to gain an intuitive understanding to be able to identify situations where the pattern is relevant.

At the root of our conceptual model for patterns is a basic `Pattern` class whose concept offers the four core elements of patterns. Context, solution and consequences are described as content in semi-structured documents.

```
model Patterns
class Pattern {
 content problem      :StructuredDocument
         solution     :StructuredDocument
         consequences :StructuredDocument
 concept characteristic name :String
         relationship   collaborators :ClassDescription
         relationship   collaboratorAspects :ClassMember }
```

Furthermore due to the availability of openness and dynamics, a CCMS allows users to model the patterns to any degree of specificity they want. This can even mean that a class is created for one specific pattern alone if this is required by the learning context. We demonstrate this with the composite pattern:

```
class CompositePattern refines Pattern {
 concept characteristic name        :String := "Composite"
         relationship   component   :ClassDescription
         relationship   composite   :ClassDescription
         relationship   leaf        :ClassDescription
         relationship   composition :AssociationDescription
         ⋮                                                    }
```

To capture pattern applications in a way meaningful to students, the applications have to be put into context of the whole application they were made in. The respective parts of the application domain can be captured by using our `GeneralSoftwareEngineering` model (see [3]). By refining the general pattern model and using the SE model, concrete patterns can be described, fig. 2 gives a brief example for the Composite pattern. Instances of this class are used to describe concrete applications of the pattern:

```
model CompositePatternApplication
from GeneralSoftwareEngineering import
 ClassDescription,Association
from Patterns import CompositePattern
let compositeApp := create CompositePattern {
 problem := ...
 ⋮
 component :=
  lookfor ClassDescription { name="Figure" package=... }
 composition :=
  lookfor Association { source=composite target=component }
 ⋮                                                          }
```

4 Pattern Learning Scenarios

The application of design patterns has become a commonly accepted design activity. Therefore, teaching the most useful patterns has become an important

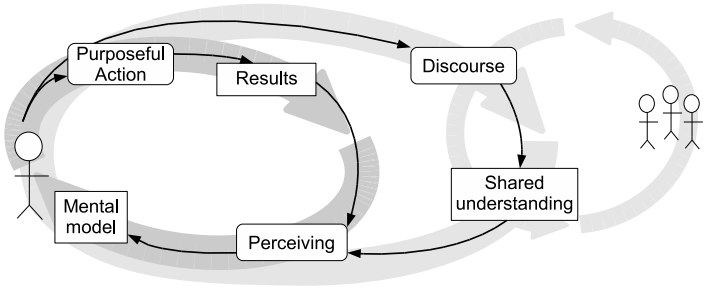

Fig. 3. Learning cycle from the learner's point of view. Inspired by [1]

part of the education or training of software developers. However, applying patterns requires tacit knowledge that cannot be studied on a purely theoretical basis. One needs to learn to actively apply patterns. In this section we argue that active learning is supported well by CCM systems.

A pattern is more than a solution to a problem in a context [24]. Especially for teaching purposes improved descriptions of design patterns are needed. Using the open modeling of the CCM approach such improved descriptions can be formulated, in particular including content. By such combined descriptions, CCM allows active learning processes for design pattern application.

4.1 Active Learning Through Open Dynamic CCM

As is witnessed by many taught courses, understanding content and applying the learnt are essential parts of learning [4]. Constructionists interpret learning as the construction of knowledge and not its absorption [19]. Practical application facilitates the lasting storage of information.

Fig. 3 depicts such an active way of learning. There are two levels, which differ in closeness to the learner as well as in speed of iteration. The inner cycle constitutes active learning. It is usually carried out by one learner alone. The outer circle describes the interaction with others, learners as well as teachers.

Typical e-learning systems support a passive way of learning: There are usually a number of ways to present lessons to learners [9]. The interaction of learner and system is frequently limited to formal testing.

All activities indicated in the learning circle in fig. 3 are covered by personalization [20]. It supports discourse through exchange of personalized content and conceptual models with a limited group of peer learners. Most importantly, users are enabled to take action in the system itself and learn through the results of their action. They can recombine artifacts to solve learning problems.

This allows an approach to e-learning in which learners can structurally rework or even extend the subject matter. For example, a lesson can start with a given partial model that the learners are to complete. In doing so, they ap-

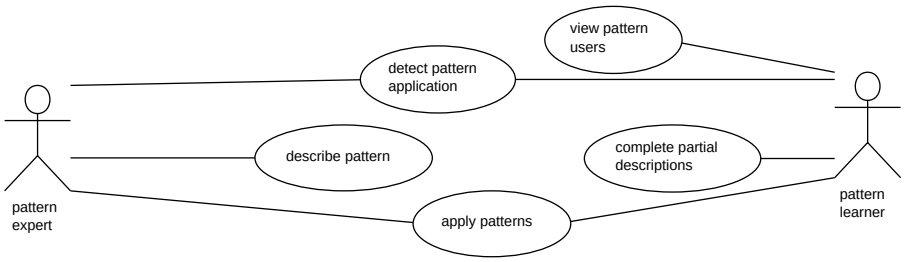

Fig. 4. Learning use cases

ply what they previously learned. Thinning out the content is again achieved through personalization (the left-out pieces are of course not deleted globally).

4.2 Requirements for Teaching Pattern-oriented Design

Teaching design patterns requires rich descriptions [24]. In sec. 3.3 asset models for typical design pattern descriptions have been shown. The openness of the model allows users to extend these for teaching purposes. Though such extensions do not automatically lead to a complete model of e-learning they can be combined with general learning models [5] to this end.

Fig. 4 shows typical use cases for a design pattern learning scenario. Here a pattern expert prepares case studies as examples or master solutions for a pattern learner, or the expert gives exercises to be solved by the learner.

In the previous section it has been mentioned that a teacher can hand partial solutions to learners. By means of personalization each learner can solve exercises individually, for instance to complete such partially given pattern descriptions or to simulate pattern applications. Personalization furthermore allows change of existing designs to try out modeling alternatives.

If only class diagrams are given, an exercise can be to detect pattern applications by capturing the classes' roles in (personal) `Pattern` asset instances.

4.3 A CCMS for Teaching Patterns

Through the CCM approach a CCMS for the management of pattern descriptions can be generated from the models presented in sec. 3.3. General requirements for such a CCMS can be deduced from [13]. The ability to serve as a tutoring system is based on the consideration of both content and conceptual models in assets and on the modeling openness.

Hosted content can serve an extensional definition of a design pattern by giving an abstract definition and by showing several applications, counter examples, etc. Conceptual models point out the design patterns that are visible in content. The dynamic nature of CCMSs permits active learning processes as discussed in the previous sections.

A design pattern CCMS can be set up as a compound system which includes the CCMS generated for `GeneralSoftwareEngineering` as a subsystem. This subsystem can then be used independently in SE processes, while the pattern description system manages the accompanying information on pattern applications.

5 Summary and Outlook

We have shown that it is beneficial for teaching purposes to model design patterns dualistically. This helps students to understand patterns by rich medial representation as well as a conceptual model. Furthermore schema personalization enables active learning as it allows learners to model the subject under study in the most appropriate way. Similar benefits arise from instance personalization where students are asked to complete partial content.

In future work it will be interesting to extend our conceptual model of design patterns to reach further into SE. It can then not only be used for teaching but will also be applicable to software creation proper. In the area of learning systems improving the reactions of the system to its users seems promising. The goal is to make learning systems react smartly to student's modeling decisions (e.g., by try to detect common misconceptions with patterns).

References

1. H. Allert, C. Richter, and W. Nejdl. Lifelong learning and second-order learning objects. *British Journal of Educational Technology*, 35(6):701–715, 2004.
2. S. Bossung, H.-W. Sehring, P. Hupe, and J. W. Schmidt. Open and dynamic schema evolution in content-intensive web applications. In *Proceedings of the Second International Conference on Web Information Systems and Technologies*, pages 109–116. INSTICC, INSTICC Press, April 2006.
3. S. Bossung, H.-W. Sehring, M. Skusa, and J. W. Schmidt. Conceptual Content Management for Software Engineering Processes. In *Advances in Databases and Information Systems: 9th East European Conference, ADBIS 2005*, volume 3631 of *Lecture Notes in Computer Science*, page 309. Springer-Verlag, 2005.
4. W. Clancy. A Tutorial on situated learning. In *Proceedings of the International Conference on Computers and Education*, 1995.
5. M. Derntl and R. Motschnig-Pitrik. Conceptual modeling of reusable learning scenarios for person-centered e-learning. In *Proceedings of International Workshop for Interactive Computer-Aided Learning (ICL'03)*. Kassel University Press, 2003.
6. G. Florijn, M. Meijers, and P. van Winsen. Tool support for object-oriented patterns. In M. Aksit and S. Matsuoka, editors, *Proceedings of ECOOP '97 - Object-Oriented Programming: 11th EuropeanConference*, volume 1241 of *Lecture Notes in Computer Science*, page 472. Springer, 1997.
7. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, NY, 1994.

8. Y.-G. Guéhéneuc, H. A. Sahraoui, and F. Zaidi. Fingerprinting Design Patterns. In *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE 2004)*, pages 172–181. IEEE Computer Society, 2004.
9. S. Guttomsen Schär and H. Krueger. Learning Technologies with Multimedia. *IEEE Multimedia*, 7(3):40–51, 2000.
10. Y.-G. Gueuc, S. Monnier, and G. Antoniol. Evaluating the Use of Design Patterns during Program Comprehension – Experimental Setting. In *Proceedings of the 1st International Workshop in Design Pattern Theory and Practice*. IEEE Computer Society Press, 2005.
11. HervAlbin-Amiot, P. Cointe, Y.-G. Gueuc, and N. Jussien. Instantiating and Detecting Design Patterns: Putting Bits and Pieces Together. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE 2001)*, pages 166–173, 2001.
12. D. Lucrio, A. Alvaro, E. S. de Almeida, and A. F. do Prado. MVCASE Tool – Working with Design Patterns. In *Proceedings of the Third Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP 2003)*, 2003.
13. G. Meszaros and J. Doble. Metapatterns: A pattern language for pattern writing. In *3rd Pattern Languages of Programming conference*, Monticello, USA, 1996.
14. T. Mikkonen. Formalizing design patterns. In *ICSE '98: Proceedings of the 20th international conference on Software engineering*, pages 115–124, Washington, DC, USA, 1998. IEEE Computer Society.
15. B.-U. Pagel and M. Winter. Towards Pattern-Based Tools. In *Proc. EuroPLoP'96*, 1996.
16. I. Reinhartz-Berger. Conceptual modeling of structure and behavior with uml the top level object-oriented framework (tloof) approach. In *Proc. ER'05*, volume 3716 of *LNCS*, pages 1–15, 2005.
17. J. W. Schmidt and H.-W. Sehring. Conceptual Content Modeling and Management: The Rationale of an Asset Language. In *Proc. Perspectives of System Informatics, PSI'03*, volume 2890 of *LNCS*, pages 469–493. Springer-Verlag, 2003.
18. J. W. Schmidt, H.-W. Sehring, M. Skusa, and A. Wienberg. Subject-Oriented Work: Lessons Learned from an Interdisciplinary Content Management Project. In *Advances in Databases and Information Systems, 5th East European Conference, ADBIS 2001*, volume 2151 of *Lecture Notes in Computer Science*, pages 3–26. Springer, September 2001.
19. U. Schroeder. Meta-Learning Functionality in eLearning Systems. In *Proc. Int. Conf. on Advances in Infrastructure for Electronic Business, Education, Science, and Medicine on the Internet*, 2002.
20. H. Sehring, S. Bossung, and J. Schmidt. Active Learning By Personalization - Lessons Learnt from Research in Conceptual Content Management. In *Proceedings of the 1st International Conference on Web Information Systems and Technologies*, pages 496–503. INSTICC Press Miami, May 2005.
21. H.-W. Sehring and J. W. Schmidt. Beyond Databases: An Asset Language for Conceptual Content Management. In *Proceedings of the 8th East European Conference on Advances in Databases and Information Systems*, volume 3255 of *LNCS*, pages 99–112. Springer-Verlag, 2004.
22. I. Sommerville. *Software Engineering.* Addison-Wesley, 2000.
23. G. Sunye, A. L. Guennec, and J.-M. Jquel. Design Patterns Application in UML. In *ECOOP*, pages 44–62, 2000.
24. J. Vlissides. *Pattern Hatching: Design Patterns Applied.* The Software Pattern Series. Addison Wesley Longman, 1998.