Didaktische Betrachtungen zur Unterrichtung von Software–Mustern im Hochschulbereich

Andreas Harrer und Markus Schneider
Gerhard-Mercator-Universität Duisburg, Technische Universität München
Lotharstr. 63, 47057 Duisburg; Boltzmannstr. 3, 85748 Garching
E-mail: harrer@collide.info, schneima@informatik.tu-muenchen.de

Abstract: Dieser Artikel beschreibt, wie Software-Muster als Unterrichtsgegenstand thematisiert werden können. Dazu geben wir zunächst eine Einführung in das Gebiet der Software-Muster und einen Überblick über Systematiken zum Umgang mit Mustern. Im Weiteren stellen wir eine Systematik vor, die – basierend auf informatischen Konzepten – besonderes Augenmerk auf die Vermittlung von Mustern im Informatik-Unterricht legt. Wir schließen mit Ausführungen zur didaktischen Aufbereitung und Vermittlung von Mustern an Hochschulen.

1 Einleitung und Intention

Vielen Informatikern, die heutzutage in selbstverständlicher Weise mit Mustern arbeiten, dürfte kaum bewusst sein, dass das Konzept „Entwurfsmuster“ ein Begriff ist, der im Kontext der Architektur von Bauwerken entwickelt wurde ([AIS79]). Die Verwendung dieses Begriffes im Rahmen der objektorientierten Modellierung etablierte sich durch [Ga95] und [Bu96]. Hierbei stellten die genannten Autoren nicht nur Muster vor, sondern präsentierten auch verschiedene Klassifikationsschemata für diese Muster.

Nachdem die objektorientierte Modellierung und somit auch das Thema „Entwurfsmuster“ bereits Eingang sowohl in Grundlagenvorlesungen der Informatik (etwa [Br01]), als auch in Spezialvorlesungen für Studenten im Hauptstudium gefunden hat (z.B. [Ha02]), stellt sich die Frage, wie das Thema „Muster“ didaktisch sinnvoll vermittelt werden kann. Zudem zeigte es sich in den genannten Vorlesungen, dass die bei [Ga95] und [Bu96] angegebenen Klassifikationsschemata nur bedingt für eine strukturierte Vermittlung dieses Themenkomplexes geeignet sind.

Die bekannten Klassifikationen von [Ga95] und [Bu96] orientieren sich in erster Linie an wichtigen Anwendungen der Software-Technik; bei der Planung einer Vorlesungs- oder Unterrichtseinheit „Entwurfsmuster“ im Rahmen des Informatikstudiums wird man sich bei der Auswahl der Muster jedoch eher an solche Muster halten, die grundlegende Konzepte der Informatik repräsentieren. Darüber hinaus ist natürlich auch die Komplexität der Muster in die Planung miteinzubeziehen. Wir benötigen also eine Klassifikation, die zum einen die informatischen Konzepte, die hinter dem jeweiligen Muster stehen, und zum anderen den Komplexitätsgrad des Musters berücksichtigt.

2 Muster in der Software-Technik

Unter einem Muster verstehen wir in diesem Zusammenhang eine abstrahierte Repräsentation erfolgreich eingesetzten Problemlösungswissens. Damit ist gemeint, dass ein Muster sich auf wiederkehrende Probleme bezieht und einen Vorschlag für eine effiziente und/oder elegante Lösung dieses Problems darstellt. Darauf basierende konkretisierte Lösungen können in der Praxis eingesetzt werden und folglich erspart die Kenntnis eines Musters, das Rad immer wieder selbst erfinden zu müssen. Ein Muster, das Probleme von Bestandteilen eines Software-Systems, wie z.B. Architektur, Subsysteme oder programmiersprachliche Aspekte, adressiert, bezeichnen wir als *Software-Muster*.

Ein Beispiel für ein solches immer wiederkehrendes Problem ist, dass häufig für die Entwicklung von Software-Systemen Bibliotheksklassen vorhanden sind, die benötigte Funktionalität bereitstellen, diese Klassen aber von der Definition ihrer Schnittstellen nicht zum restlichen Entwurf des Software-Systems passen. Eine Anpassung der Bibliotheksklasse selbst ist im Allgemeinen nicht möglich, weil der Quellcode nicht vorliegt und auch andere Systeme mit diesen Klassen arbeiten könnten. Eine Anpassung der Systemschnittstellen ist ebenso nicht erwünscht.

Eine Lösung für dieses Problem ist die Definition eines so genannten **Adapters** [Ga95], der die beiden Schnittstellen miteinander verträglich macht. Dieser Adapter implementiert auf der einen Seite die Dienstschnittstelle und verwendet auf der anderen Seite die Funktionalität der Bibliotheksklasse wieder (hier als Objektadapter durch Komposition und Delegation, alternativ ist der Klassenadapter mit Implementierungsvererbung). Innerhalb der Implementierung des Schnittstellendienstes wird dann der Dienst der Bibliotheksklasse aufgerufen. Eine graphische Darstellung dieser Grundidee findet sich im UML-Klassendiagramm in Abb. 1.

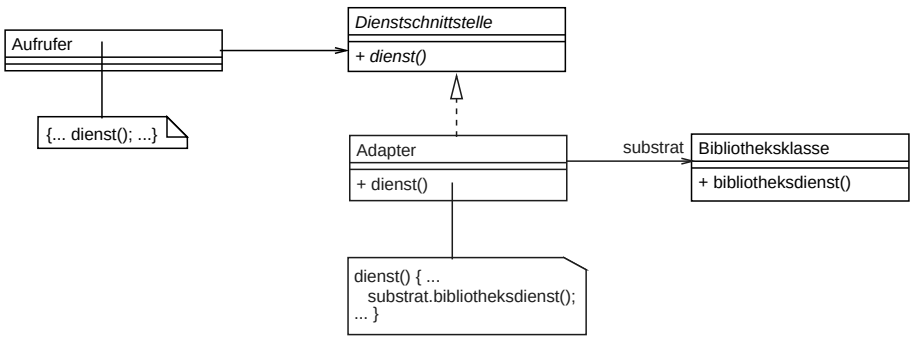

Abbildung 1: Lösung für Benutzung von Bibliotheksklassen: Adapter-Muster

Diese Grundidee kann also eingesetzt werden, wenn Bibliotheksklassen verwendet werden sollen, die von ihrer Schnittstelle her nicht unmittelbar integrierbar sind. In einem konkreten Problemfall spricht man bei Einsatz dieser Lösungsidee von einer *Instantiierung* des Musters.

3 Systematiken für Muster – Stand der Forschung und Kritik

Betrachtet man Muster im Zusammenhang miteinander, so können Software-Systeme entwickelt werden, bei denen nicht nur einzelne Teilprobleme durch den Einsatz eines bestimmten Musters gelöst werden können, sondern insgesamt ein flexibler und stimmiger Gesamtentwurf realisiert wird. Zu diesem Zweck ist es notwendig, Muster systematisch zu erfassen, zu strukturieren und Beziehungen zwischen Mustern herauszuarbeiten.

Ein erstes Hilfsmittel dazu ist die Einführung eines *Schemas* zur internen Strukturierung eines Musters. Um die Lesbarkeit von Mustern zu erhöhen und einzelne gedankliche Abschnitte explizit zu trennen, wird ein Muster in Sektionen, wie Name, Kontext, Problem, Lösung, Varianten, Implementierungsvorschlag usw. eingeteilt. Durch Verwendung eines Schemas werden Muster einheitlich beschrieben und damit besser vergleichbar. Die Pionierarbeit im Bereich von Mustern (in der Architektur) [AIS79] verwendet ein Schema, das die Elemente Name, Kontext, Problem, Lösung und Fortsetzungsmuster beinhaltet. Da für Software-Muster etwas andere Akzente wesentlich sind, wie z.B. die Implementierbarkeit oder eine Beschreibung von Wechselwirkungen zwischen Komponenten, werden üblicherweise für Software-Muster Schemata verwendet, die diesen Akzenten Rechnung tragen. Sowohl in [Ga95] als auch in [Bu96] werden zur näheren Illustration der Lösung Beispiele und Implementierungsvorschläge in das Schema integriert und die Lösung selbst mit strukturellen und dynamischen Elementen beschrieben.

Eine Strukturierung nicht nur einzelner Muster, sondern des gesamten Musterkatalogs hat den Vorteil, dass dadurch Muster gruppiert werden können (z.B. nach Gemeinsamkeiten, Einsatzzweck), gezielt nach Mustern gesucht und deren Eignung für bestimmte Problemstellungen verglichen werden kann. Zu diesem Zweck werden Muster durch eine *Klassifizierung* eingeteilt. Diese kann auch anhand mehrerer Kategorien durchgeführt werden.

In [Ga95] wird eine Klassifizierung gemäß zweier Dimensionen vorgenommen:

- **Bereich** (*scope*), der sich darauf bezieht, ob die Lösung des Musters sich mehr auf Beziehungen zwischen Objekten (im einführenden Beispiel in Form des Objektadapters) oder zwischen Klassen (ebenda der Klassenadapter) konzentriert.
- **Zweck** (*purpose*), der unterschieden wird in „mit Erzeugung befasst“ (*creational*, z.B. Fabrikmethode-Muster), struktur-orientiert (*structural*, z.B. Kompositum-Muster) und verhaltens-orientiert (*behavioral*, z.B. Beobachter-Muster).

In [Bu96] findet ebenfalls eine zweidimensionale Klassifizierung statt:

- **Abstraktionsgrad** des Musters, der angibt, ob das Muster sich auf Gesamtsysteme bezieht (Architekturmuster), auf Subsysteme (Entwurfsmuster) oder einzelne Implementierungsprobleme in bestimmten Programmiersprachen (Idiom).
- **Problemkategorie** des Musters, die beschreibt, in welchem Kontext das Muster eingesetzt werden kann und welches Problemgebiet das Muster adressiert, beispielsweise verteilte Systeme, interaktive Systeme oder Kommunikation.

Bei der Bewertung, wie sich diese Klassifizierungen für die Unterrichtung von Mustern eignen, mussten wir feststellen, dass beide den Aspekt der geeigneten Auswahl und Vermittlung von Mustern nur zum Teil abdecken. Die Klassifizierung in [Ga95] unterscheidet recht grob in 2*3 Kategorien, was zur Folge hat, dass Muster, die relativ wenige, zum Teil lediglich strukturelle aber nicht inhaltliche, Gemeinsamkeiten aufweisen oder ganz unterschiedliche informatische Konzepte ausdrücken (siehe Abschnitt 4), in dieselbe Kategorie eingeteilt werden. Die Klassifizierung in [Bu96] weist mit dem Abstraktionsgrad eine Dimension auf, die für die Unterrichtung von Mustern gut geeignet ist, um eine Grobeinteilung der Muster vorzunehmen. In der Dimension Problemkategorie werden jedoch Systemklassen, wie verteilte oder interaktive Systeme, mit Konzepten, wie Kommunikation oder Zugriffsregelung, vermischt, die Klassifizierung verliert an dieser Stelle an Exaktheit. Eine Systematik, die auf die Unterrichtung von Mustern zugeschnitten ist, stellen wir in Abschnitt 4 vor.

4 Entwicklung einer konzeptuell-didaktischen Systematik

4.1 Konzeptuelle Klassifizierungskategorien

Um bei der Unterrichtung von Mustern nicht nur Detailwissen zu vermitteln, sondern auch Zusammenhangs- und Überblickswissen zu erzeugen, schlagen wir für die Systematik von Mustern folgenden Ansatz vor: Wesentlich ist die klare Darstellung, welche informatischen Grundkonzepte den einzelnen Mustern innewohnen, um Gemeinsamkeiten und andere Beziehungen zwischen den Mustern besser herausarbeiten zu können.

Klassifizierungskategorien für Muster
Im Folgenden leiten wir die Klassifizierungskategorien für Muster aus allgemeinen Prinzipien des objektorientierten Softwareentwurfs her. Dazu betrachten wir zunächst grob den Prozess objektorientierter Modellierung:

Verwendung von Architekturmustern: Im Rahmen des Entwurfs eines objektorientierten Modells tritt zunächst eine Phase der Modularisierung auf: Das System wird grob in interagierende Objekte zerlegt und es werden geeignete Klassen eingeführt; die Modellierung der Interaktion macht im Allgemeinen die Einführung spezieller Klassen zur Steuerung der Interaktion erforderlich.

Verwendung von Entwurfsmustern: Anschließend wird das Modell verfeinert, d.h. es werden Attribute und Methoden der Objekte im Detail modelliert; hierbei ist in der Regel eine weitere Zerlegung bisher eingeführter Objekte erforderlich, da beispielsweise Methoden modularisiert werden müssen und es nun aus Gründen der Flexibilität und Wiederverwendung sinnvoll ist, Hilfsmethoden in eigenständige Objekte auszulagern. Werden hierbei Klassenbibliotheken verwendet, so kann sich wiederum die Notwendigkeit ergeben, die Interaktion der Klassen durch weitere Klassen zu steuern, z.B. durch einen Adapter.

Verwaltung der Objekte: In dieser letzten Phase werden implementierungsnahe Probleme im Zusammenhang mit der Verwaltung von Objekten, z.B. Erzeugung, sequentieller Zugriff auf aggregierte Objekte, modelliert.

Der objektorientierte Modellierungsprozess ist also offensichtlich ein iterativer Prozess, dessen Ablauf von den Konzepten **„Modularisierung“**, gegebenenfalls **„Wiederverwendung“** vorhandener Klassen und Modellierung der **„Interaktion“** dominiert wird. Am Ende der Modellierung steht eine Phase, die zentral das **„Management“**, also die Verwaltung von Objekten, behandelt. Die Mehrzahl der bei [Ga95] genannten Entwurfsmuster lässt sich nun schwerpunktmäßig einem der genannten vier Konzepte zuordnen; im Folgenden diskutieren wir diese Zuordnung und geben, sofern möglich, übergeordnete Muster zur graphischen Beschreibung des jeweiligen Konzeptes an.

Modellierung der Interaktion
Die Muster, die primär diesem Konzept zugeordnet werden können, weisen eine Komponente auf, die interagierende Objekte entkoppelt und als zentrales Element die gesamte Interaktion organisiert. Muster, die in erster Linie Standardlösungen zur Modellierung der Interaktion darstellen, sind Mediator, Beobachter, Proxy, Fassade, Adapter – der Adapter repräsentiert jedoch auch in hohem Maße das Konzept der Wiederverwendung –, sowie das Muster der Zuständigkeitskette. Das Muster, das die Gemeinsamkeiten der genannten Standardlösungen am besten erfasst, zeigt Abb. 2. In den jeweiligen Mustern ergeben sich Unterschiede dadurch, in welcher Art und Weise diese Komponenten miteinander interagieren, z.B. werden alle Empfänger benachrichtigt oder genau einer usw.

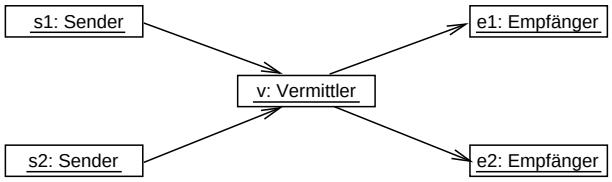

Abbildung 2: Übergeordnetes Muster: Interaktion

Dekomposition
Hier betrachten wir Muster, die die Bearbeitung komplizierter Sachverhalte in einzelne Teilkonzepte bzw. Aspekte zerlegen und die Bearbeitung in isolierte Repräsentationen auslagern. Derartige Probleme können algorithmischer Natur sein, Zustandsabhängigkeit repräsentieren oder nach Abstraktionsgrad separieren. Diesem Dekompositionskonzept können die Muster Strategie, Zustand, Brücke, Befehl, sowie Schablonenmethode zugeordnet werden. Vergleicht man die genannten Muster, so zeigt sich, dass bei der Schablonenmethode die Bearbeitung in Unterklassen vorgenommen wird, bei den übrigen Mustern durch Aggregation und Delegation. Abstrahiert man von diesen verschiedenen Mustern auf ein übergeordnetes Muster, so ergibt sich das in Abb. 3 skizzierte Muster.

Spezialfall: Rekursive Dekomposition
Einen für die Informatik besonders wichtigen Spezialfall der Dekomposition erhalten wir, wenn sich der zu modellierende Sachverhalt rekursiv zerlegen und strukturieren lässt. Das Muster, das das Konzept der Rekursion auf Objektebene beschreibt, ist das Kompositum-Muster. Das Klassendiagramm des Kompositum-Musters lässt sich aus Abb. 3 erzeugen, indem man das Hauptkonzept mit dem konkreten Nebenkonzept identifiziert und die Viel-

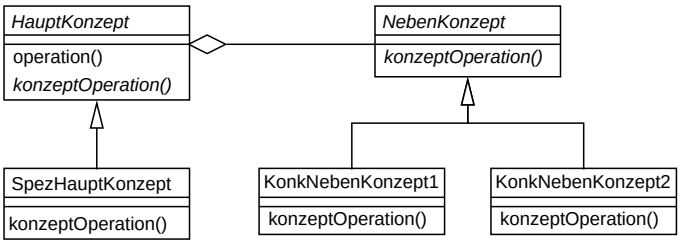

Abbildung 3: Übergeordnetes Muster: Dekomposition

fachheit der Aggregation modifiziert. Weitere Muster mit rekursiver Modularisierung sind das Dekorator-Muster und das Interpreter-Muster.

Wiederverwendung
Dieses Konzept beschreibt die Wiederverwendung von Eigenschaften von Systemen, Klassen oder Objekten, um Mehrfacherstellung von Modellen oder Code zu vermeiden. Es wird in idealer Weise durch das in Abschnitt 2 beschriebene Adapter-Muster repräsentiert.

Management
Unter diesem Begriff fassen wir Muster zusammen, die implementierungsnahe Probleme der Verwaltung von Objekten modellieren: Das sind etwa die Muster zur Erzeugung von Objekten, d.h. Muster, die bei [Ga95] als Creational Patterns bezeichnet werden. Ebenso rechnen wir Muster, wie das Iterator-Muster, das den Zugriff auf sequentiell durchlaufbare Objekte (Bäume, Listen) behandelt, zu dieser Gruppe.

4.2 Klassifizierung nach Komplexitätsaspekten

Bei der Unterrichtung von Mustern ist ein weiterer wichtiger Gesichtspunkt, wie komplex die Vermittlung und der Themengegenstand selbst sind. Um dem Dozenten bei der geeigneten Auswahl von Mustern zu helfen, damit das Niveau des Gegenstands dem Kenntnisstand der Schüler auch angemessen ist, fügen wir unserer Systematik eine weitere Dimension hinzu, die für jedes Muster angibt, wie komplex die Vermittlung des Musters ist.

Komplexität von Mustern:

- **Schwierigkeit des Verständnisses:** Ist das Muster in seiner Gesamtheit einfach oder schwer zu verstehen?
- **Umfang der Diskussionspunkte:** Gibt es viele Punkte zur Diskussion, zur Abwägung bezüglich des Einsatzes und der Auswirkungen, die es mit sich bringt?
- **Variationsmöglichkeiten:** Gibt es verschiedene Lösungsmöglichkeiten oder Variationen im Kontext?

4.3 Weitere Klassifizierungskategorien

Neben diesen beiden, hauptsächlich an didaktischen Gesichtspunkten orientierten, Dimensionen einer Klassifizierung, wollen wir unsere Klassifizierung noch – in leicht modifizierter Form – um Aspekte aus der Klassifizierung in [Bu96] ergänzen: Die dortige Einteilung nach dem *Abstraktionsgrad* in Architekturmuster, Entwurfsmuster und Idiome liefert eine gute Grobeinteilung für die **Art** des beschriebenen Musters. Statt der Dimension der *Problemkategorie* definieren wir eine Dimension **Einsatzgebiet**, die ausdrückt, ob ein Muster allgemein einsetzbar ist oder für ganz spezielle Systemtypen und Umgebungen charakteristisch ist (z.B. verteilte Systeme, interaktive Systeme). Dadurch haben wir in unserer Klassifizierung Systemklassen und informatische Konzepte sauber getrennt, wohingegen in [Bu96] eine Vermischung dieser Aspekte vorliegt.

4.4 Beispiel der Klassifizierung

Um die Verwendbarkeit unserer Klassifizierung aufzuzeigen, geben wir in Tabelle 1 einen Überblick, wie darin die in [Ga95] beschriebenen Entwurfsmuster klassifiziert werden.

5 Unterrichtung von Mustern in der Hochschule

5.1 Unterrichtsplanung – Auswahl von Mustern

Bei der Thematisierung von Mustern in der Informatikausbildung an Hochschulen ist die Wahl von Anspruchsniveau und Grundlagenwissen der Hörer wesentlicher Gesichtspunkt.

Erste Erfahrungen der Unterrichtung von Mustern in größeren Veranstaltungen sammelten wir in der Grundvorlesung im Studienjahr 2000/2001: Dort wurden einige Muster vorgestellt, um typische Lösungen im Software-Entwurf anzusprechen, beispielsweise wurde die hierarchische Zerlegung eines Gesamtsystems in Subsysteme mit Hilfe des Kompositum-Entwurfsmusters erklärt. Die Auswahl der konkreten Muster erfolgte hierbei hauptsächlich aufgrund pragmatischer Gesichtspunkte, dass bestimmte Muster mit in der Vorlesung vorgestellten Inhalten gut harmonierten, beispielsweise wurde im Rahmen der ereignisbasierten Programmierung das Beobachter-Entwurfsmuster und das Model-View-Controller-Architekturmuster (MVC) aufgegriffen.

In der Vorlesung „Muster in der Software-Technik“ im Sommersemester 2002 wurden aus der Fülle von Mustern, die mittlerweile in der Literatur zu finden sind, zunächst Architekturmuster [Bu96] ausgewählt, die ein breites Spektrum an Anwendungsgebieten abdecken, um den Hörern die Relevanz und Einsetzbarkeit von Mustern in weiten Bereichen der Informatik aufzuzeigen. Damit beschränkten wir uns also nicht nur auf die Software-Technik im Engeren, sondern behandelten auch Muster im Kontext von Betriebssystemen (Mikrokern), verteilten Systemen (Broker) und Systemen mit Benutzerinteraktion

Name	Informatisches Konzept	Komplexität
Abstrakte Fabrik	Management	**
Erbauer	Management	***
Fabrikmethode	Management	*
Prototyp	Management	*
Singleton	Management	*
Adapter	Wiederverwendung, Interaktionsregelung	*
Brücke	Dekomposition, Wiederverwendung	***
Dekorator	Rekursive Dekomposition	**
Fassade	Interaktionsregelung	*
Fliegengewicht	Management	**
Kompositum	Rekursive Dekomposition	**
Proxy	Interaktionsregelung, Management	**
Befehl	Interaktionsregelung, Dekomposition	*
Beobachter	Interaktionsregelung	**
Besucher	Management, Wiederverwendung	**
Interpreter	Rekursive Dekomposition	**
Iterator	Management	*
Mediator	Interaktionsregelung, Dekomposition	**
Memento	Management	**
Schablonenmethode	Dekomposition, Wiederverwendung	*
Strategie	Dekomposition	**
Zustand	Dekomposition	**
Zuständigkeitskette	Interaktionsregelung, Dekomposition	**

Tabelle 1: Klassifizierung der Muster nach [Ga95] mit der konzeptuellen Systematik; von oben nach unten sind in Gruppen creational, structural und behavioral patterns aufgelistet.

(MVC). Bei der Auswahl der weniger abstrakten Entwurfsmuster stellten wir insbesondere Zusammenhänge zur Anwendbarkeit deren Lösungen im größeren Zusammenhang der davor thematisierten Architekturmuster her, z.B. eines Fassade-Entwurfsmusters für Mikrokern-Architekuren oder eines Beobachter-Entwurfsmusters für MVC. Unsere Absicht war es, durch dieses Vorgehen das Zusammenhangswissen und die Kombinierbarkeit von verschiedenen Mustern bei der Software-Entwicklung zu betonen.

5.2 Unterrichtsdurchführung – Vermittlung von Mustern

Die Thematik der Muster in der Software-Technik eignet sich sehr gut für die Kombination verschiedener didaktischer Vorgehensweisen. In [Ha02] beschrieben und verwendeten wir diesen kombinierten Ansatz in einer Vorlesung zu dieser Thematik:

Induktives Vorgehen: Im Bereich der Software-Entwicklung bietet sich dieses Vorgehen besonders an, um am Beispiel eines Programmcodes konkret auf das dahinterliegende Konzept hinzuweisen. In [Ha02] wenden wir dieses Vorgehen für Erzeugungsmuster an,

um an konkreten Programmausschnitten die Problemstellung eines Musters zu verdeutlichen. Dieses stärker codeorientierte Vorgehen harmoniert mit der *muster-integrierenden Software-Entwicklung*, bei der Schritt für Schritt verhaltens-bewahrende Programmtransformationen, so genannte *Refaktorierungsschritte* [Fo99], durchgeführt werden, bis die Ideen eines Musters in das System integriert sind. Durch Abstraktion von der Code-Ebene gelangt man zur allgemeinen Lösung, die von dem integrierten Muster verkörpert wird.

Deduktives Vorgehen: Dieses Vorgehen wird unmittelbar durch die Form unseres Muster-Schemas unterstützt, das zunächst allgemeine Grundgedanken der Einsetzbarkeit in Form des Kontexts, der Problematik und der wiederverwendbaren Lösungsidee aufzeigt und diese allgemeinen Gesichtspunkte an einem darauffolgenden Beispiel konkretisiert. In der Software-Entwicklung findet sich ein Pendant zum deduktiven Vorgehen in so genannten „top-down"-Ansätzen, in denen vom Abstrakteren hin zum Konkreten geschritten wird. Dieses Prinzip wird – bezogen auf die Software-Entwicklung mit Mustern – in der *muster-orientierten Software-Entwicklung* [Ha02] verfolgt, in der vom abstrakten Grundentwurf in Form eines Architekturmusters, wie z.B. Model-View-Controller, welches dann instantiiert wird, durch Verfeinerung mit Mustern niedrigeren Abstraktionsgrads, wie z.B. Beobachter-Entwurfsmuster, ein Software-System erstellt wird.

Spiralförmiges Vorgehen: In diesem Ansatz [Be01] wird ein Konzept in mehreren Iterationen schrittweise um Details angereichert. Zunächst werden die essentiellen Elemente des Lehrinhalts vermittelt und erst darauffolgend nach und nach Ergänzungen vorgenommen, bis alle Aspekte diskutiert wurden. Dieses Vorgehen erleichtert das Verständnis, insbesondere der essentiellen Punkte, da in keinem Schritt bei den Studierenden eine zu große kognitive Last aufgebaut wird. Insbesondere für komplexere Architekturmuster, wie z.B. eine Broker-Architektur [Bu96], eignet sich dieses Vorgehen. Ein Vorlesungsbeispiel dazu findet sich in [Ha02].

Eine Kombination dieser drei Vorgehensweisen wurde im Rahmen einer Vorlesung zum Thema „Muster in der Software-Technik" im Sommersemester 2002 an der Technischen Universität eingesetzt. In den darauf folgenden Prüfungen konnten wir feststellen, dass die verschiedenen Vorgehensweisen den Studenten so vertraut waren, dass sie in Transfer-Aufgaben z.B. mit Mustern, die deduktiv vorgestellt worden waren, auch mit induktiver Vorgehensweise umgehen konnten, indem sie aus einem Problembeispiel die allgemeine Lösung ableiten konnten. Eine formale Evaluation war zu diesem Zeitpunkt leider nicht möglich, ist aber für zukünftige Lehrveranstaltungen dieser Art geplant.

6 Zusammenfassung und Ausblick

In der vorliegenden Arbeit befassten wir uns kritisch mit Klassifikationsschemata für Software-Muster. Ausgangspunkt unserer Betrachtungen waren Lehrerfahrungen im Bereich der Entwurfsmuster. Dabei zeigte es sich, dass aus didaktischer Sicht eine differenzierteres Klassifikationsschema als die bisher bekannten erforderlich ist.

Die hier vorgeschlagene Klassifizierung weist als entscheidende Neuerung eine *konzeptuelle* Kategorie auf, d.h wir identifizieren die Grundkonzepte objektorientierter Model-

lierung, die die einzelnen Muster repräsentieren. Dies sind: **„Interaktion“** von Objekten, **„Dekomposition“** eines Systems in Bestandteile und **„Wiederverwendung“** von Klassen. Spezielle, implementierungsnahe Muster zur Verwaltung von Objekten ordneten wir der Kategorie **„Management“** zu. Alle der bei [Ga95] beschriebenen Entwurfsmuster sind mindestens einer dieser vier Kategorien zuzurechnen.

Darüber hinaus führten wir die didaktische Klassifizierungskategorie **Komplexität** ein, um insbesondere bei der Planung von Vorlesungen und Seminaren, die Auswahl eines geeigneten Musters zu erleichtern. Von der bekannten Klassifikation gemäß [Bu96] wurden die Kategorien *Abstraktionsgrad* und *Einsatzgebiet* weitgehend beibehalten.

Zum Abschluss dieser Arbeit stellten wir unsere Erfahrungen mit Vorlesungen im Grund- und Hauptstudium zum Thema Software-Muster dar.

Die hier dargelegten Untersuchungen könnten jedoch nicht nur für den Hochschul-, sondern auch für den Schulbereich (Sekundarstufe II) Bedeutung erlangen: Nach dem derzeitigen Stand der Dinge wird die objektorientierte Modellierung ein wichtiges Themengebiet des Pflichtfaches „Informatik“ an bayerischen Gymnasien sein [Hu02]. Da Entwurfsmuster bewährte Standardlösungen für Probleme im Bereich der objektorientierten Modellierung darstellen, stellt sich die Frage, ob die Thematisierung von Entwurfsmustern in vereinfachter Form nicht auch im Schulbereich sinnvoll wäre. Die hier vorgestellte konzeptuelle Klassifizierung könnte dabei hilfreich sein; denn nur solche Muster, die wesentliche Konzepte der Informatik auf einem für die Schule geeigneten Komplexitätsniveau repräsentieren, könnten in der Schule effektiv verwendet werden.

Literaturverzeichnis

[AIS79] Alexander, C.; Ishikawa, S.; Silverstein, M.: The Timeless Way of Building, volume 1 of Center for Environmental Structure Series. Oxford University Press, New York, 1979.

[Be01] Bergin, J.: Fourteen Pedagogical Patterns. Pattern language description, Pace University, 2001. http://hillside.net/patterns/.

[Br01] Brügge B.: Einführung in die Informatik I und II. Vorlesung 2000/2001, Technische Universität München, http://wwwbruegge.in.tum.de/teaching/ss01/Info2/

[Bu96] Buschmann F.; Meunier R.; Rohnert, H.; Sommerlad, P.; Stal, M.: A System of Patterns. John Wiley & Sons, Chichester, 1996.

[Fo99] Fowler, M.; Beck, K.; Brant, J.; Opdyke, W.; Roberts, D.: Refactoring: Improving the Design of Existing Code. Addison-Wesley, Reading, MA, 1999.

[Ga95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1995.

[Ha02] Harrer, A.: Muster in der Software–Technik. Vorlesung Sommersemester 2002, Technische Universität München, http://wwwbruegge.in.tum.de/teaching/ss02/muster/

[Hu02] Hubwieser, P.: Object Models of IT-Systems supporting Cognitive Structures in Novice Courses of Informatics. SECIII, Dortmund, 2002