Informatik-Spektrum 19:326–327 (1996)

*Das aktuelle Schlagwort**

Entwurfsmuster

Klaus Quibeldey-Cirkel

Der Architekt Christopher Alexander hält die *Keynote*-Ansprache auf der OOPSLA '96 – der bedeutendsten Konferenz über die objektorientierte Softwaretechnik. Sein Musterbegriff, entwickelt in den 70er Jahren für die Gebäude-Architektur und Städteplanung [1], hat das Schlagwort *Entwurfsmuster* (Design Patterns) geprägt. Es war wohl die Suche nach der Konzeption eines *Architektur-Handbuchs* für den Software-Entwurf (OOPSLA '92), die zur alexandrinischen *Musterform* führte. 1995 hieß das bestverkaufte Buch der Informatik „Design Patterns: Elements of Reusable Object-Oriented Software" [6]. Der Musterbegriff hat eine Bewegung ausgelöst, die ihresgleichen sucht[1]. Internationale Workshops finden bereits turnusmäßig statt: Pattern Languages of Program Design [4], PLoP '94–'96, EuroPLoP '96.

Alltagssprachlich verstehen wir unter einem Muster: eine *Vorlage*, nach der etwas hergestellt wird, ein beispielhaftes *Vorbild* oder eine regelmäßige, sich wiederholende *Struktur*. In allen drei Interpretationen wird der Begriff in der (objektorientierten) Softwaretechnik angewandt: *konstruktiv* als Vorlage (generische Entwurfsmuster), *deskriptiv* als Vorbild für die Dokumentation von Entwurfsentscheidungen und -kompromissen[2] und *strukturell* als Orientierung in einem komplexen Entwurf. Entwurfsmuster sind praxisbewährte Lösungsstrukturen für stereotype Entwurfsprobleme. Im Gegensatz zu Halbfabrikaten wie Klassenbibliotheken und Frameworks, werden *immaterielle* Entwürfe wiederverwendbar. Die handlungsorientierte Musterform macht das *Erfahrungswissen* der Experten leichter zugänglich. Zur Disposition des Entwerfers stehen nicht mehr nur einzelne Klassen, sondern vielmehr Konfigurationen mehrerer Klassen – *Mikro-Architekturen*.

Beispiel: Der Entwurf einer grafischen Oberfläche wäre ohne die Orientierung an Erfahrungswerten mühsam, ad hoc und fehlerträchtig. Ein *mustergestützter* Entwurf könnte nun folgendermaßen verlaufen: Wir definieren zunächst unser konkretes Problem (z. B. die Entkopplung unterschiedlicher, aber konsistenter Visualisierungen desselben Datenbestands) und suchen dann in einem *Musterbuch* nach geeigneten Vorlagen. Wir finden ein *strategisches* Muster namens „Model-View-Controller" (MVC, Abb. 1) und erfahren, daß es sich in interaktiven Grafikanwendungen bewährt hat. Auf wenigen Seiten werden dort das immer wiederkehrende *Entwurfsproblem*, der *Anwendungskontext* und eine *Lösungsstruktur* beschrieben.

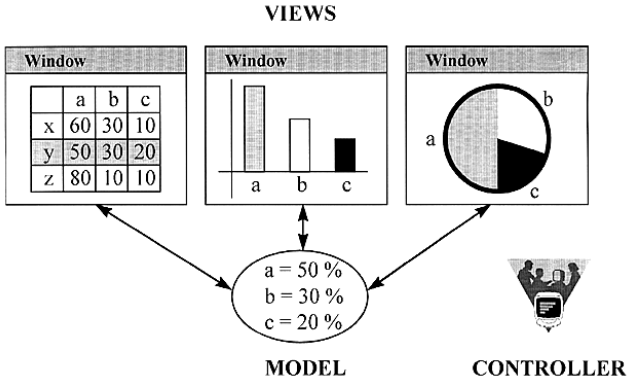

Abb. 1 Illustration zum MVC-Muster [6]

Die grafischen und textuellen Erläuterungen sind anschaulich und intuitiv. Wir erfahren, wie die Dreiteilung[3] die ehernen Entwurfsprinzipien der Softwaretechnik umsetzt (*Information hiding* von Parnas und *schwache Kopplung zwischen und starke Kohäsion in den Modulen* von Simon [7]). Dazu werden wir auf feinkörnigere *taktische* Muster verwiesen, die immer nach den gleichen Kriterien beschrieben sind: Mustername und Synonyme, Einordnung, Zweck und Einsatzmotive, Anwendungs-Szenarien, allgemeine Lösungsstruktur, beteiligte Klassen und deren Zusammenspiel, Vor- und Nachteile, Beispiele und Nachweise über den erfolgreichen Praxiseinsatz, Querverweise und Abgrenzungen zu ähnlichen Mustern.

Ein solches Taktikmuster heißt „Publisher-Subscriber" (Klassendiagramm in Abb. 2). Es beschreibt die Korrelation zwischen dem zentralen Datenbestand (model) und den verschiedenen Sichten (views). Das „Publisher"-Objekt unterrichtet alle „Subscriber"-Objekte über Zustandsänderungen (NotifySubscribers()), indem es deren Aktualisierung (Update()) veranlaßt. Dabei sind die „Subscriber"-Objekte untereinander *entkoppelt* und vollständig austauschbar (*dynamisch* zur Programmlaufzeit). Das Muster beschreibt also eine *1-zu-n*-Abhängigkeit zwischen Objekten. So läßt sich zum Beispiel ein zentraler Datenbestand (in der Publisher-Rolle) auf verteilten Rechnern oder in mehreren „Fenstern" desselben Rechners unterschiedlich darstellen: als Kuchen-, Balkendiagramme oder Tabellen (Subscriber-Rollen).

Viele der Muster, wie Publisher-Subscriber, stammen aus GUI-Frameworks (ET++, HotDraw, InterViews). Es waren er-

[1] http://st-www.cs.uiuc.edu/users/patterns/patterns.html

[2] http://www.ti.et-inf.uni-siegen.de/Entwurfsmuster

Klaus Quibeldey-Cirkel
Universität Siegen, Technische Informatik, Hölderlinstrasse 3,
D-57068 Siegen, Tel.: 0271/7403210, Fax: 0271/7403344,
e-mail: quibeldey@ti.et-inf.uni-siegen.de

* Vorschläge an: Prof. Dr. Frank Puppe, Institut für Informatik, Universität Würzburg, Am Hubland, D-97047 Würzburg
und Dieter Steinbauer, GEZ, Freimersdorfer Weg 6, D-50829 Köln

[3] Der Controller für die Abfrage und Steuerung der Benutzeraktionen ist für unser Beispiel nicht relevant.

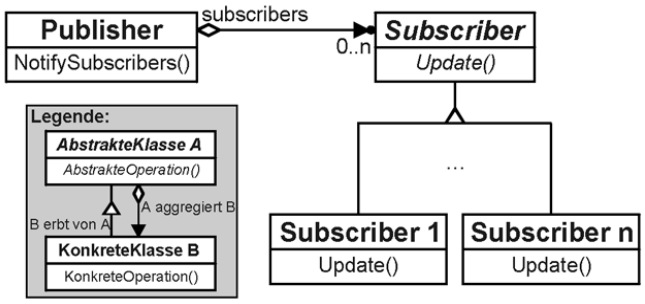

Abb. 2 Publisher-Subscriber-Muster

fahrene Framework-Entwickler (Autoren von [6]), die die Musterform favorisierten. Muster zur Framework-Dokumentation legen das Erfahrungswissen der Entwickler offen und verringern so den Lern- und Einarbeitungsaufwand für den Framework-Anwender. Softwaremuster sind allgemein kein originär akademisches Thema, vielmehr gehen sie aus der industriellen Praxis hervor. Immer mehr Softwarehäuser versuchen, ihren langjährigen Erfahrungsfundus in (hausinternen) Musterbüchern zu dokumentieren. Derzeit publizierte Musterbücher unterscheiden sich hauptsächlich in ihrem *Abstraktionsgrad*. So gibt es Muster für die Systemanalyse [5], für den Programmentwurf [6] und für die Codierung [3].

Das thematische Spektrum der Entwurfsmuster legt eine Bestandsaufnahme in komprimierter Form nahe: **ETHOS** [7].
E wie „economic“: Bekanntlich liegt das Potential der Objektorientierung in der Wiederverwendung. Die *Kapselung* von Daten und Operationen erlaubt die systematische Wiederverwendung von Struktur und Verhalten in *einer* Komponente, der Klasse. Entwurfsmuster erweitern nun die Wiederverwendung auf *kommunizierende Gruppen* von Klassen. Wiederverwendung findet „im großen“ statt, das heißt auf einer *architektonischen* Ebene, ähnlich der eines Frameworks. Im Gegensatz dazu sind Entwurfsmuster aber abstrakt (kein Code) und somit anwendungsunabhängig. Sie sind deshalb gut geeignet, ein Framework zu dokumentieren.
T wie „technical“: Die Form eines Softwaremusters wird derzeit noch kontrovers diskutiert. Die *alexandrinische* Urform, das Triplett aus Problem, Kontext und Lösung, ist *narrativ*, von epischer Breite. Die „Portland“-Softwaremuster halten diese Form weitestgehend ein[4]. Die „Gamma-Muster“ sind schematischer, sie werden nach 13 Kriterien beschrieben (wie in unserem Beispiel). Nach der Granularität der Lösungsstruktur lassen sich zwei Kategorien unterscheiden: Sprachspezifische Codierungs-Muster stellen die kleinste Kategorie: *Idiome* [3] (syntaktische Strukturen). Gamma-Muster dagegen sind grobkörniger (Strukturen aus 3 – 4 Klassen), sie unterstützen den allgemeinen Programmentwurf.

Das ideale Medium für Entwurfsmuster ist Hypertext, da ein Muster nicht isoliert auftritt, sondern stets assoziativ im Verbund, und die Navigation im Informationsraum Voraussetzung für deren effiziente Verwendung ist. Alexander spricht hier von Mustersprachen (treffender wäre wohl Begriffssysteme); Buschmann et al. sprechen von Mustersystemen [2]. Musterbücher werden schon bald ihre Gutenberg-Form überwinden; Proprietäre Entwurfsmuster (also solche, die firmenspezifisches Know-how widerspiegeln) werden in Intranetzen, nichtproprietäre im Internet verfügbar gehalten.

H wie „human“: *Lern*-psychologisch sind Entwurfsmuster Problem-Lösungspaare gleicher Struktur, die primär auf den Erwerb von Fertigkeiten ausgerichtet sind. Sie werden bereits als didaktisches Vehikel in der industriellen Schulung und der akademischen Lehre eingesetzt[5]. *Motivations*-psychologisch steigern Entwurfsmuster das Selbstwertgefühl des Programmierers: Orientiert er sich an den Entwurfsmustern eines Experten, dokumentiert er damit die Qualität seines Programms und distanziert sich zugleich vom „Hackertum“.
O wie „organizational“: Derzeit sind Bemühungen im Gange, die bewährten Prinzipien und Strategien der Projektführung und der Teamorganisation in Musterform zu beschreiben [4]. Das Wissen über gute Organisationsformen entbehrt bisher der pragmatischen Vermittlung. Organisationsmuster versprechen hier Abhilfe.
S wie „social“: Mit Hilfe der Entwurfsmuster wird ein gemeinsames *Vokabular* geschaffen, das eine effiziente Kommunikation unter den Entwerfern erlaubt und so die Diskussion über komplexe Zusammenhänge erleichtert. Originalton aus einem Projekt-Meeting: „Nehmen wir an dieser Stelle das Publisher-Subscriber-Muster, um die Komponenten zu entkoppeln.“ Für *sprachkundige* Entwerfer, die das gleiche Mustervokabular beherrschen, ist damit alles gesagt!
Gesamtbewertung: Die literarische Gattung *Musterbuch* kann auf eine lange Tradition verweisen. Sie hat ihre Anfänge nicht erst bei Christopher Alexander. Für die Entwicklung des Handwerks und der Industrie im 19. Jahrhundert waren Musterbücher von grundlegender Bedeutung. Sie stellten ein Kompendium handwerklichen Wissens dar. In ihnen wurden die über die Jahrhunderte entwickelten und gesammelten Erfahrungen festgeschrieben. Die heutigen Handwerke entwickeln sich wesentlich schneller und unterliegen viel stärker dem Wandel der Technik. Auch der Software-Entwurf wurde immer schon als *Handwerkskunst* verstanden – im guten Sinne als *Kunsthandwerk*: „Computer Programming as an Art“ (Knuth), im schlechten Sinne als *Spezialistentum*: kryptische Programme, die nur der „Künstler“ selbst warten kann. Mit Hilfe von Musterbüchern à la Gamma et al. können wir fortan die *gute* Kunst des Software-Handwerks beschreiben und vor allem *lehren*.

Literatur

1. Alexander, C. et al.: A Pattern Language: Towns, Buildings, Construction. New York: Oxford University Press 1977
2. Buschmann, F. et al.: Pattern-Oriented Software Architecture: A System of Patterns. Chichester: Wiley & Sons 1996
3. Coplien, J. O.: Advanced C++ Programming Styles and Idioms. Reading MA: Addison-Wesley 1991
4. Coplien, J. O.; Schmidt, D. C. (Hrsg.): Pattern Languages of Program Design. Reading MA: AddisonWesley 1995
5. Fowler, M.: Analysis Patterns: Reusable Object Models. Reading MA: Addison-Wesley 1996
6. Gamma, E. et al.: Design Patterns: Elements of Reusable Object-Oriented Software. Reading MA: Addison-Wesley 1995, 5. Druck
7. Quibeldey-Cirkel, K.: Das Objekt-Paradigma in der Informatik. Stuttgart: Teubner 1994

Eingegangen am 15.07.1996, in überarbeiteter Form am 24.10.1996

[4] http://c2.com/ppr/about/portland.html

[5] http://st-www.cs.uiuc.edu/users/patterns/Education.html