

Geschichte, Gegenwart und Zukunft von Software-Patterns

20 Jahre später

Michael Stal



Muster gibt es in vielen Zusammenhängen, von der Schneiderei bis hin zur Architektur. Dass sie sich auch im Softwaredesign als nützlich erweisen können, haben Informatiker bereits vor 20 Jahren entdeckt. Heute haben Design-Patterns nicht nur einen festen Platz in der Softwareentwicklung, sondern werden auch auf immer neue Anforderungen zugeschnitten.

Ende der Siebzigerjahre veröffentlichte Christopher Alexander die Bücher „A Pattern Language“ und „A Timeless Way of Building“ [1], die das Konzept von Entwurfsmustern beziehungsweise Pattern-Sprachen einführen. Ein Pattern repräsentiert demzufolge eine be-

währte architektonische Lösung für eine häufig wiederkehrende Aufgabe, weshalb Alexander nicht ohne Grund von „Quality without a Name“ spricht. Sein zweiter Band sollte sechzehn Jahre später zu der Pattern-Bewegung in der Software-Engineering-Community führen.

Im Jahr 1993 luden Grady Booch und Kent Beck mehrere Protagonisten aus der objektorientierten Szene zu einem Treffen ein, um ein gemeinsames Fundament für Software-Patterns zu schaffen. Veranstaltungsort war eine Berghütte in Colorado. Eines der Themen war die Fusion der Ideen Christopher Alexanders mit jüngeren Arbeiten von Erich Gamma als Basis für die Entwurfsmuster. Die Gründung einer Non-Profit-Organisation mit Fokus auf Software-Patterns, die den naheliegenden Namen Hillside Group erhielt, war ein weiteres Ergebnis des Retreats. Bis heute spielt diese Gruppe eine zentrale Rolle in der Pattern-Community. Aus ihr gingen zahlreiche Konferenzen zum Thema hervor, die sogenannten PLoPs (Pattern Languages of Programming).

Viele Jahre nach dem Erscheinen seiner Bücher hielt Christopher Alexander auf der Konferenz OOPSLA 1996 im kalifornischen San Jose zur Begeisterung der Teilnehmer eine Keynote über Muster – teilweise gab es glühende Anhänger seiner Lehren. Das alles ist insofern bemerkenswert, als Alexander ein Gebäudearchitekt ist und kein Informatiker. Trotzdem hat er die Softwareentwicklung wohl mehr beeinflusst als die eigene Disziplin. (Zum Keynote-Video bei YouTube und weiteren Ressourcen im Web siehe „Alle Links“ im blauen Balken am Ende des Artikels.)

Back to the Future

Springen wir aber noch einmal ein paar Jahre zurück. Anfang der Neunzigerjahre schrieben zwei Teams parallel an Pattern-Büchern. Zum einen die sogenannte Gang of Four (GoF) mit Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides und zum anderen das POSA-Team (POSA = Pattern-Oriented Software Architecture) bei Siemens [2]. Als Mitautor des 1996 erschienenen Buches „Pattern-Oriented Software Architecture – Volume 1: A System of Patterns“ [3] hatte ich das Glück, aktiv dabei gewesen zu sein. Die beiden Autorentäms pflegten damals regen Kontakt miteinander, um keine Patterns zu duplizieren.

Als 1995 das Buch „Design Patterns: Elements of Reusable Object-Oriented Software“ der Gang of Four erschien, kam alles anders als vermutet. Den Autoren hatte Jim Coplien prophezeit, dass Pattern-Autoren bald überall als VIPs betrachtet würden, was ich selbst für völlig abwegig hielt. Doch das Buch von Erich, Ralph, Richard und John schlug ein wie

eine Bombe und gilt zurecht als Bibel nicht nur der Pattern-Community.

Das ein Jahr später erschienene erste von fünf POSA-Büchern führte zusätzlich zu Design-Patterns Architekturmuster, Idiome und Pattern-Systeme ein. Es verkaufte sich ebenfalls hervorragend und erhielt gleich im darauffolgenden Jahr 1997 den Jolt Productivity Award.

Interessanterweise kamen zumindest anfangs alle Autoren aus dem Umfeld der Software-Frameworks, aus denen letztlich viele der dokumentierten Muster stammen. Bei genauer Überlegung ist das nicht weiter überraschend, da Frameworks die Nutzung bewährter wiederverwendbarer Konzepte geradezu bedingen.

Objektorientierung als treibender Faktor

Um den Erfolg von Patterns zu verstehen, sollte man sich in die Zeit ihres Auf-tauchens zurückversetzen. Als ihren „Siegeszug“ begannen, war Objektorientierung zwar schon in aller Munde, aber noch nicht überall verbreitet, geschweige denn verstanden. Unter den OO-Programmiersprachen galt C++ als führend, Smalltalk als Hoffnungsträger und Java als neuer Stern am Horizont, während C# noch nicht das Licht der Welt erblickt hatte. Die drei Amigos Booch, Jacobsen und Rumbaugh schraubten fleißig an der UML-Spezifikation.

Nun liefern zwar Sprachen und Methoden das notwendige Werkzeug zum Lösen von Entwurfsproblemen, lösen selbst aber keine. Patterns hingegen geben konkrete Hilfestellungen zu konkreten Problemen und unterstützen die Wiederverwendung von Best Practices für das Design von Anwendungen.

Der Untertitel des Design-Pattern-Buchs der GoF, „Elements of Reusable Object-Oriented Software“, ist daher zwar aus damaliger Sicht verständlich, aus heutiger erscheint die Einschränkung auf



Die Bibel unter den Design-Pattern-Büchern, verfasst von der sogenannten Gang of Four – im (und am) Auto von links nach rechts: John Vlissides, Richard Helm, Ralph Johnson, Erich Gamma (Abb. 1)

By Purpose

	Creational	Structural	Behavioral
By Scope	Class	• Factory Method • Adapter (class)	• Interpreter • Template Method
	Object	• Abstract Factory • Builder • Composite • Prototype • Singleton	• Chain of Responsibility • Command • Iterator • Mediator • Memento • Observer • State • Strategy • Visitor

Patterns lassen sich anhand diverser Kriterien kategorisieren. Im Buch der GoF gibt es zum Beispiel eine Unterscheidung danach, worauf sich das Muster hauptsächlich auswirkt – auf Struktur, Verhalten oder Objekterzeugung.

Objektorientierung allerdings unnötig. Design-Wiederverwendung ist nicht abhängig von Entwurfsparadigmen und lässt sich auch bei strukturiertem oder funktionalem Entwurf gut einsetzen. Das Proxy-Muster beispielsweise hat seinen Ursprung in der strukturierten Programmierung mit C. Monaden sind Entwurfsmuster aus der Welt der funktionalen Programmierung.

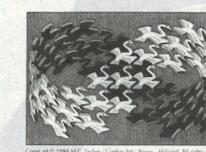
TRACT

- Vor 20 Jahren erschien als Trendsetter das Buch „Design Patterns: Elements of Reusable Object-Oriented Design“.
- Vater der Pattern-Bewegung ist der Gebäudearchitekt Christopher Alexander mit seinem Buch „A Timeless Way of Building“.
- Patterns haben sich inzwischen als ein Standardwerkzeug der Softwareentwickler etabliert.
- Das Finden, Dokumentieren und Anwenden von Patterns erfordert ein systematisches, qualitätsorientiertes Vorgehen.

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Cover art © 1994 McGraw-Hill Book Company, Inc. All rights reserved.

Foreword by Grady Booch

Software-Patterns bestehen aber nicht nur aus Design-Patterns. Stattdessen gibt es verschiedene Geschmacksrichtungen von Mustern, die sich hauptsächlich in ihrer Granularität unterscheiden:

Idiome befinden sich auf dem Level von Programmiersprachen oder APIs. Sie beinhalten Best Practices zum Umgang mit der betreffenden Sprache oder API. Etwa das berühmte C++-Idiom, das empfiehlt, im Konstruktor Ressourcen zu akquirieren und sie im Destruktor wieder freizugeben, um durch die Automatismen der Laufzeitumgebung Ressourcenlecks zu vermeiden.

Design-Patterns sollen im Allgemeinen Entwurfsprobleme auf Komponentenebene lösen, etwa das Strategy-Muster, das den Nutzer einer Schnittstelle von der gewählten Implementierung unabhängig macht.

Design-Taktiken bewegen sich auf derselben Abstraktionsebene wie Design-Patterns, konzentrieren sich aber auf nicht funktionale Eigenschaften wie die Perfor-

manz. Beispiele hierfür sind unter anderem Caching, Lazy Acquisition und Thread Pools.

Architektur-Patterns liefern grundlegende strategische Entwurfsentscheidungen, die die komplette Architektur oder überwiegende Teile davon betreffen. Beispiele: Layers, Microkernel und Pipes & Filters.

Das Eintauchen in Design-Patterns und Idiome verhalf Entwicklern zu intensiven Einsichten in objektorientierte Me-

chanismen. Durch das gezielte Anwenden solcher Muster und Design-Taktiken kam es darüber hinaus zu einem spürbaren Produktivitätsschub. Und dem Thema Softwarearchitektur wurde aufgrund der Architektur-Patterns mehr Aufmerksamkeit geschenkt.

Doch was genau ist eigentlich ein Pattern? Und was ist es nicht? Laut Brian Foote repräsentiert ein Pattern die völlige Ignoranz von Originalität. Das soll heißen, dass Muster bewährte Lösungen häu-

fig wiederkehrender Probleme darstellen und keine bislang unentdeckten Entwurfsperlen sind. Also das glatte Gegenteil des Star-Trek-Mantras: „to boldly go where no man has gone before.“

Fehlinterpretationen

Von einem Informatiker habe ich einst bei einem Projekt zu hören bekommen, er interessiere sich nicht für Patterns,

weil sie sich durch ein wenig Nachdenken ganz einfach herleiten ließen. Im Lauf des Projekts hat er seine eigene These selbst gründlich widerlegt und seine Meinung revidiert. Muster haben einen hohen intellektuellen und praktischen Wert. Das Aufspüren der ihnen innerwohnenden Lösungsansätze ist alles andere als trivial. Oder andersherum ausgedrückt: Ein leicht herzuleitender Lösungsansatz für ein Problem ist kein Pattern.

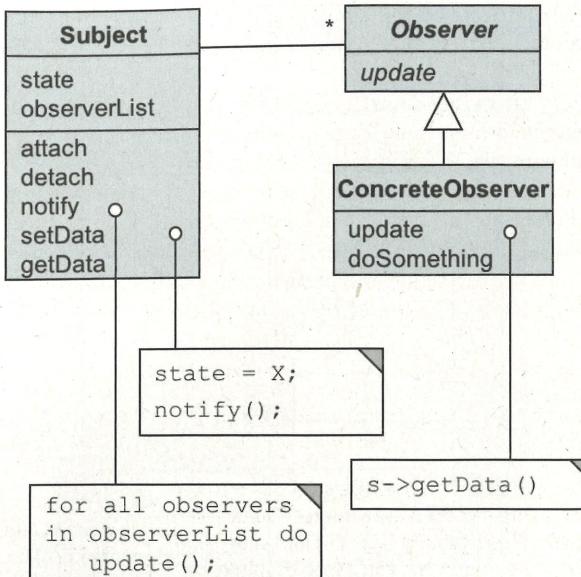
Es existieren noch andere Fehlinterpretationen. Auf einer Konferenz in der Wiener Hofburg, ganz im historischen Rahmen, nahmen Erich (Gamma) und ich einst an einer Paneldiskussion über Patterns teil. Ein Zuschauer meldete sich zu Wort und verkündete mit stolzgeschwellter Brust, dass er alle Muster aus dem GoF-Buch in seiner Software verwendet habe. Er erntete keine Anerkennung, sondern stieß auf unser Unverständnis. Moral von der Geschichte: Das Hammer-and-Nail-Prinzip lässt sich auch hervorragend mit Mustern in die Tat umsetzen. Nebenbei bemerkt: Wenn der Einsatz von Mustern überlegt und systematisch erfolgt, kann eine hohe Pattern-Dichte durchaus ein Indikator für hohe Qualität sein.

Die Inflation der AntiPatterns

Im Jahr 1998 erschien das Buch „AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis“ von Brown et al. [4]. Trotz seines kommerziellen Erfolgs stieß es in der Community auf wenig Gegenliebe. AntiPatterns drehen den Spieß um, indem sie mögliche Stolperfallen als Basis verwenden, statt sich auf bewährte Lösungen zu konzentrieren. Anders ausgedrückt widmen sie sich der Frage, warum bestimmte Entwurfs-„Lösungen“ immer wieder zu Komplikationen führen. Das Problem an AntiPatterns liegt in der Tatsache begründet, dass es natürlich wesentlich mehr Möglichkeiten gibt, einen falschen Lösungsweg einzuschlagen als einen richtigen. Daher kommt es zu einer Inflation von AntiPatterns, von denen viele sehr allgemein oder banal erscheinen. Die meisten erweisen sich daher schlicht als wenig nützlich bis unbrauchbar.

In Gartners Hype Cycle gibt es bekanntlich den sogenannten „Peak of inflated Expectations“. Das ist der Punkt, an dem die Erwartungen an eine Technologie ihren Gipfel erreichen. Die Patternbewegung hatte in den Neunzigern diesen steilen Gipfel in Rekordzeit erklimmen. Man-

Das Observer Pattern führt als Verantwortlichkeiten die Rolle des Subject (beobachtetes Objekt) und den Observer (Beobachter) ein (Abb. 3).



che Manager erhofften sich von Patterns, ganze Architekturen per Mausklick generieren zu können. Einige Softwarehersteller wollten Bibliotheken mit generischen Pattern-Implementierungen bauen. Andere glaubten, die gesamte Softwareentwicklung komplett mit Patterns abdecken zu können. Ganz abgesehen von zum Glück vergeblichen Versuchen, Patterns über Softwarepatente schützen zu lassen.

In meiner beruflichen Laufbahn haben Muster nicht nur bewährte Lösungen für wiederkehrende Probleme in einem speziellen Kontext angeboten. Ein Beispiel zeigt der Kasten „Leaky Bucket Counter“. Vielmehr hat sich in vielen anderen architektonischen Zusammenhängen das Wissen darüber bewährt, wie Pattern-Beschreibungen strukturiert an Probleme herangehen.

Pattern-Beschreibung und Pattern-Form

Zunächst geben sie der Lösung des Problems einen **Namen**, etwa Observer Pattern (GoF). Es ist leichter, vom Observer anstatt ständig über die komplexe architektonische Struktur der beteiligten Komponenten zu sprechen. Damit erhält die „Quality without a Name“ einen Namen.

Darüber hinaus definieren Pattern-Beschreibungen den **Kontext**, in dem die zu lösende Aufgabe liegt. Zum Beispiel: „Ergebnisse in einer Komponente an andere interessierte Komponenten melden.“

Danach beschreiben sie die **Aufgabe** selbst sowie die Anforderungen (Forces) an die Lösung. Für das Observer Pattern hieße das: Ergebnisse in einer Komponente sollen Beobachter auf Wunsch erhalten

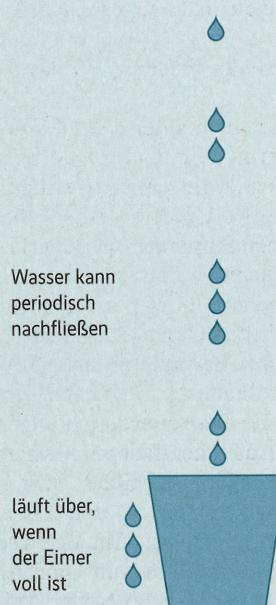
Darüber hinaus spezifizieren Autoren die Dynamik anhand typischer Szenarien im Lösungsraum. Ein Beispiel: Eine beobachtete Komponente (Subject) informiert beim Eintreffen eines Ereignisses durch die Liste der Beobachter und ruft deren update()-Methoden auf. Als Parameter übergibt die Komponente Informationen über das betreffende Ereignis (Abbildung 4).

Im **Implementierungsteil** enthalten Patterns einen (nicht notwendigerweise vollständigen) Mikroprozess mit den wesentlichen Schritten der Pattern-Anwendung. Da diese Muster als Blaupausen einen generischen Ansatz verfolgen, kann gerade bei Architektur-Patterns die Implementierung sehr umfangreich und komplex ausfallen. Dieser Abschnitt ist folglich der aufwendigste bei der Dokumentation der Entwurfsmuster. Beim Observer wäre zu spezifizieren, wie eine

Dieser Kasten stellt als Beispiel das weniger bekannte Pattern „Leaky Bucket Counter“ vor. Seine Beschreibung liegt in einer sogenannten Patternform vor, von denen es mittlerweile eine ganze Reihe gibt.

Name
Leaky Bucket Counter Pattern

Autor
James Coplien



Das Pattern „Leaky Bucket Counter“ soll verhindern, dass bei der Datenübertragung über Streams einzelne Pakete verloren gehen (Abb. 2).

Leaky Bucket Counter

Kontext

Abgrenzen temporärer Probleme von permanenten Fehlern bei der Übertragung von Datenströmen.

Problem

Bei der Datenübertragung über Streams können gelegentlich einzelne Pakete verloren gehen oder fehlerhaft sein. Bei einigen Systemen gelten derartige Probleme als hinzunehmende temporäre Ereignisse, sofern ihre Anzahl nicht eine definierte Grenze übersteigt. Wie sieht für solche Konstellationen ein geeignetes Fehlermanagementkonzept aus, das temporäre von permanenten Fehlern unterscheiden kann?

Lösung (Statik und Dynamik)

Es gibt die folgenden Rollen:

- der Sender des Datenstroms;
- der Empfänger des Datenstroms;
- die atomare Einheit der Übertragung (Bild, Ton, Paket, ...);
- die eigentliche Streaming-Komponente.

Vorausgesetzt wird eine getaktete Übertragung, zum Beispiel ein 1-Sekunden-Takt. Jede pro Takt gesendete Einheit repräsentiert eine konstante Wassermenge von v_{cycle} , die beim Empfänger in einem Auffangbehälter landet.

Das Gefäß hat ein Leck, aus dem alle in Takte eine Wassermenge v_{leak} entweicht. Das Auslaufen von Wasser soll selbstverständlich nur dann möglich sein, wenn das Gefäß nicht leer ist.

Für jede nicht oder verfälscht ankommende Übertragungseinheit füllt der Empfänger v_{error} ins Gefäß.

Am Gefäß befindet sich in einer vorher festgelegten Höhe (das heißt bei einer bestimmten maximalen Füllstandsmenge) ein Schwellwert v_{max} . Erreicht der Wasserstand $v_{current}$ diesen Wert, gibt das System Alarm, da ein permanenter Fehler anzunehmen ist.

Implementierung

Zunächst ist der Takt zu bestimmen. Bei einer Videoübertragung mit 60 fps lautet er dementsprechend 1/60 sec.

Statt des Wassergefäßes kommt ein ganzzahliger Counter $n_{current}$ zum Einsatz. Bei Ankunft einer Übertragungseinheit erfolgt ein Inkrementieren des Zählers um n_{cycle} , bei Fehlern die Erhöhung um einen Faktor n_{error} . Nach m Takten wird der Zähler um n_{leak} dekrementiert. Bei Erreichen von n_{max} liegt ein permanenter Fehler vor.

Varianten

Das Wassergefäß ist eine Warteschlange mit einer Maximalkapazität. Sobald die Warteschlange diese erreicht, verwirft sie alle eintrifftenden Pakete, bis sie wieder mit dem Einangsprozess synchronisiert ist.

Konsequenzen

Pro

- Der Implementierungsaufwand ist vergleichsweise gering.
- Die gewählte Metapher eignet sich sehr gut für die Fehlerbehandlung beim Streaming.
- Die Operationen im Leaky Bucket Counter sind in der Regel minimalinvasiv, wenn man sie mit der benötigten Übertragungszeit vergleicht.
- Durch Variieren der verschiedenen Größen lassen sich Fehlererkennung und -behandlung justieren beziehungsweise optimieren.

Kontra

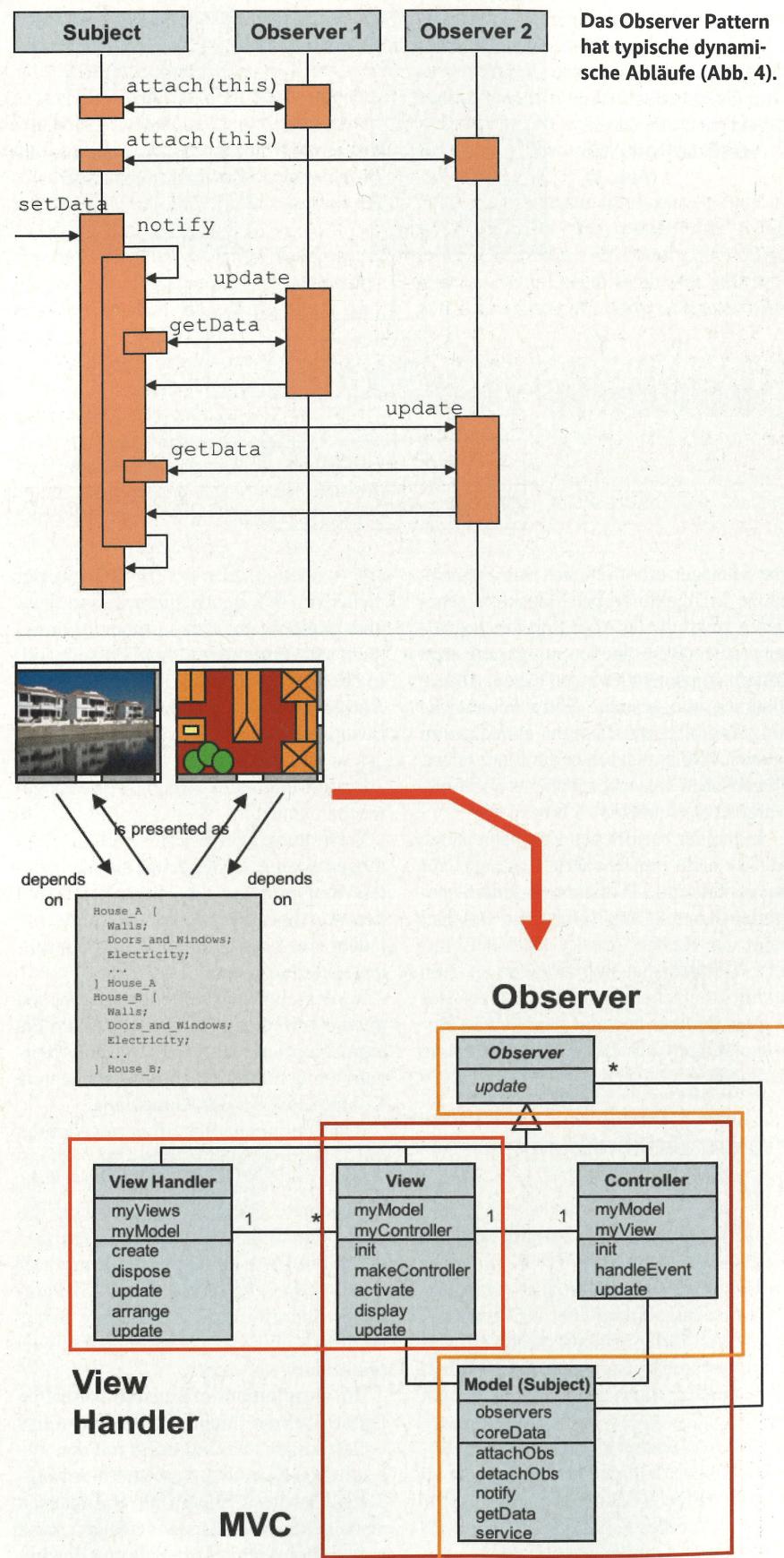
- Das Pattern eignet sich nicht für zuverlässige Systeme, bei denen Fehler möglichst früh erkannt werden müssen.
- Die Konfiguration des Patterns benötigt unter Umständen empirische Analysen.

Bekannte Anwendungen

- Telekommunikationsanwendungen
- Multimedia, A/V-Streaming

Siehe auch

Nutzung des Strategy Pattern, um die Implementierung vor der eigentlichen Anwendung zu verbergen.



Drei in einem: Patterns sind keine Inseln, sondern in den Entwurf integriert und auch oft miteinander verzahnt. Das Modell in diesem Szenario ist gleichzeitig fachliche Datenbank der Gebäudeelemente, Model im Model-View-Controller Architekturmuster und Subject im Observer-Pattern (Abb. 5).

es Views über Änderungen im Model unterrichtet.

Da Patterns als Blaupausen für den Entwurf fungieren, bieten sie viele Freiheitsgrade bei der Umsetzung beziehungsweise der Integration ins eigene System. Entwickler müssen also überlegen, wie sie Patterns einsetzen möchten, und dabei in mehreren Schritten vorgehen.

Mehrere Schritte zum richtigen Pattern

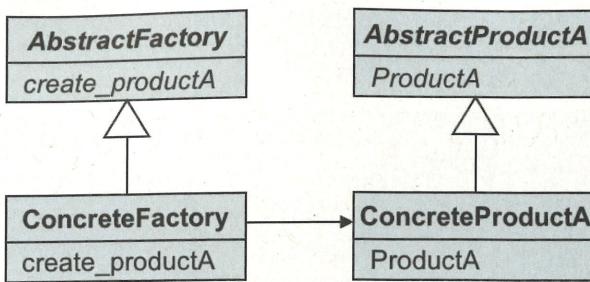
Suche: Stößt die Architektin oder Entwicklerin auf ein Problem, das sich gut abstrahieren und verallgemeinern lässt, sollte sie sich zunächst auf die Suche nach passenden Mustern begeben. Dazu bietet sich neben dem Literaturstudium die Onlinesuche an. Das Zauberwort für die Suchmaschine heißt „Pattern Catalog“. Eine gute Anlaufadresse ist auch die Website der Hillside Group. Ausschlaggebend bei der Suche sind der für ein Pattern dokumentierte Kontext sowie die Problembeschreibung inklusive der Anforderungen.

Analyse und Eingrenzung: Wird sie fündig, durchforstet sie die Beschreibungen der gefundenen Muster nach Voraussetzungen für deren Einsatz und insbesondere nach den Konsequenzen ihrer Anwendung. Nur wenn diese Punkte mit den eigenen Anforderungen übereinstimmen, steht das Pattern zur Wahl.

Selektion: Danach selektiert die Entwicklerin das für ihren Zweck am besten geeignete Entwurfsmuster.

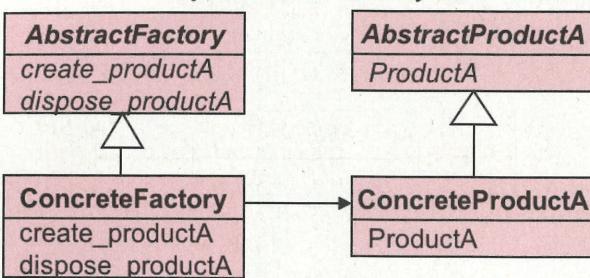
Implementierung: Bei der Integration des Musters ins eigene Design orientiert

Das Abstract Factory Pattern in seiner Originalfassung ist aus Symmetriegründen unvollständig (Abb. 6).



Abstract Factory 2002 (Kevlin Henney):

- object creation and deletion methods
- experience: if object creation is complex, object deletion is likely too



che Rollen in verschiedenen Mustern innehaben (Abbildung 5).

Patterns sind nicht für die Ewigkeit ausgelegt oder unveränderlich. Sie geben nur den augenblicklichen Stand des Wissens und der Erfahrungen wieder. Beispielsweise betrachtet ein Großteil der Community das Singleton Pattern [2] heute nicht mehr als sinnvolles Muster. Andere wie Null Object Pattern oder Extension Object beziehungsweise Extension Interface wären dafür vielversprechende Kandidaten für eine Pattern-Bibel 2.0. Wieder andere benötigen eine Anpassung. Ein häufig zitiertes Beispiel in diesem Zusammenhang ist das Abstract Factory Pattern ([2], Abbildung 6).

PowerShell 4.0 / 5.0 Seminare

Einführung für System- und Netzwerkadministratoren

Die Windows PowerShell (WPS) ist mittlerweile etabliert in der Windows-Systemadministration. Microsoft und andere Anbieter stellen viele Tausend Befehle für Konsolen- und Skriptbasierte Administration bereit. Einige Aufgaben lassen sich inzwischen gar nicht mehr per GUI, sondern nur noch per PowerShell lösen.

In diesem Kurs lernen Sie von drei bekannten PowerShell-Experten und -Buchautoren die Basiskonzepte der PowerShell sowie zahlreiche Einsatzgebiete an vielen Praxisbeispielen kennen. Sie üben selbst an der PowerShell-Konsole, der PowerShell-Skriptumgebung „ISE“ und kostenfreien Zusatzwerkzeugen.

Referenten:

Dr. Tobias Weltner,
Dr. Holger Schwichtenberg,
Thomas Wiefel

Seminar-Termine:

20. - 23. September 2016, Essen
18. - 21. Oktober 2016, München
15. - 18. November 2016, Essen
29.11.- 02.12.2016, München

Buchen
Sie jetzt!

In Zusammenarbeit mit:

www.IT-Visions.de®
Dr. Holger Schwichtenberg

Weitere Infos unter: www.powershell-schulungen.de



Writer-Workshop auf der VikingPLoP. Die PLoP-Konferenzen widmen sich ausschließlich dem Thema Patterns (PLoP = Pattern Languages of Programs) (Abb. 7).

In seiner ursprünglichen Patternbeschreibung führt Abstract Factory eine *create()*-Methode ein, die die Komplexität der Objekterstellung des dahinter liegenden Typs verbergen soll. Aus Symmetriegründen stellt sich aber die Frage, warum es keine *dispose()*-Methode gibt, da auch das Zerstören solcher Objekte meistens komplex ist. In einem refaktorisierten Pattern gäbe es daher neben *create()*- auch *dispose()*-Methoden.

Auf der Suche nach neuen Mustern

Doch woher kommen Patterns eigentlich? Gilt für sie der berühmte Spruch: „Das Kunstwerk war bereits vorhanden; ich musste es nur noch vom Stein befreien“? Im Grunde lautet die Antwort Ja und Nein. Wie oben erwähnt, enthalten Patterns Expertenwissen darüber, wie sich häufig wiederkehrende Aufgaben auf intelligente Weise lösen lassen. Für diesen Lösungsansatz fordert die Pattern-Community konkrete Beweise. Erst wenn der Ansatz nachweislich in mindestens drei voneinander unabhängigen Produktivsystemen zur Anwendung kam, haben die Autoren es tatsächlich mit einem Pattern-Kandidaten zu tun.

Ich selbst war Anfang der Neunzigerjahre in Sachen Architekturen und Middleware für verteilte Systeme unterwegs. Nachdem ein Kollege aus der Automatisierungstechnik vorschlug, mangels vorhandener Dokumentation die Architek-

tur von Microsofts Middleware – also COM, OLE, DCOM – zu analysieren und zu beschreiben, machte ich mich als OMG-CORBA-Anhänger daran, diese konkurrierenden Technologien eingehend zu durchleuchten. Dabei trat zu Tage, dass nicht nur Microsoft und die OMG, sondern auch viele andere Hersteller und Standardisierungsgremien im Grunde immer wieder konzeptionell dieselben Lösungen für ähnliche Problemstellungen nutzen. Auf diese Weise erblickten Patterns wie Broker oder Proxy [POSA Vol. 1] das Licht der Welt.

Das Identifizieren von Entwurfsmustern ist der erste Schritt. Als zweiter kommt die Beschreibung des Problems sowie seines Kontextes, auf die das Pattern in spe zielt. Der darauffolgende dritte und schwerste Schritt ist die Variability-Commonality-Analyse. Dabei stellen sich folgende Fragen:

- Welche Teile in der Lösung bleiben über mehrere Anwendungen invariant, welche sind fallabhängig und damit variabel?
- Wie gestaltet sich die häufigste Verwendung des Lösungsansatzes und was sind wichtige Varianten?
- Enthält das potentielle Pattern weitere Muster?
- Welches sind die notwendigen Implementierungsschritte?
- Welche Vorteile und Nachteile hat das Pattern?
- Gibt es verwandte Muster für eine ähnliche Aufgabe?

Sollte jemand das Angebot erhalten, an einem Pattern-Buch mitzuschreiben, über-

legt er es sich vor einer Zusage besser zweimal. Da Patterns wiederverwendbare Blaupausen für Architektur und Entwurf darstellen, müssen sie eine extrem hohe Qualität haben. Daher ist es essenziell, die Beschreibung eines Musters gründlich und wiederholt von vielen Experten prüfen zu lassen. Dies bedeutet insbesondere, dass ein Pattern-Autor die Beschreibung sehr oft über den Haufen wirft und gänzlich neu formuliert.

Workshops mit konstruktiven Diskussionen

Als wichtige Review-Methodik haben sich einständige Writer-Workshops etabliert, die wie folgt ablaufen: Zwingende Voraussetzung für den Workshop ist das vorherige intensive Studium des Dokuments durch die Teilnehmer. Der Autor nimmt in einem Writer-Workshop eine eher passive Rolle ein und verfolgt stattdessen die Diskussion. Am Workshop beteiligt sind ein Moderator, der Autor und die Reviewer.

Zu Beginn liest der Pattern-Verfasser eine aus seiner Sicht wichtige Passage seines Dokuments vor. Danach beschreiben zwei gut vorbereitete Reviewer das Muster aus ihrer Sicht. Im nächsten Schritt identifizieren die beteiligten Reviewer die Stärken des Dokuments in Bezug auf Inhalt und Form. Nun erfolgt eine konstruktive Diskussion darüber, was sich aus Reviewer-Sicht verbessern ließe. Der Autor hat anschließend die Gelegenheit, Rückfragen zu stellen. Dabei soll es um Klärung gehen, nicht um Verteidigung des Dokuments. Am Schluss dankt die Runde dem Autor für das Erstellen der Pattern-Beschreibung.

Ein Pattern ist keine Insel

Nach meiner Erfahrung sind Writer-Workshops effiziente Werkzeuge für Reviews, weil sie an einigen Stellen auch in die Tiefe gehen, statt überall nur an der Oberfläche zu kratzen. Daher eignet sich diese Art von Arbeitskreisen auch für Design-Reviews. Selbstredend gibt es Muster für das effektive Durchführen von Writer-Workshops.

Wie das Beispiel Model-View-Controller weiter oben zeigt, bilden Patterns keine einsamen Inseln, sondern können miteinander in Beziehung stehen. Das Buch der GoF definiert einen Katalog, der voneinander unabhängige Patterns für ganz unterschiedliche Problemstellungen enthält. Es gibt also in einem solchen Ka-

talog keine Abhängigkeiten der Muster untereinander.

Die POSA-Bücher 1, 2 und 3 enthalten voneinander unabhängige Muster für parallele und verteilte Systeme sowie für das Ressourcenmanagement. Beispielsweise benötigt das Architekturmuster Broker weitere Design-Patterns zu seiner Implementierung wie Proxy, Forwarder-Receiver oder Client-Dispatcher-Server. Diese Entwurfsmuster gehen also eine stärkere Verbindung miteinander ein. Es handelt sich um ein Pattern-System.

Im extremen Fall lässt sich eine gesamte Domäne durch eine Menge tief gehend und komplex miteinander verwobener Patterns abdecken. Dafür hat sich die Bezeichnung Pattern-Sprache eingebürgert.

Was war, was ist und was sein wird

Nach 20 Jahren ist es ein guter Zeitpunkt, sowohl über die Geschichte der Pattern-Technik als auch über deren Gegenwart und Zukunft zu reflektieren. Die Geschichte hat der Artikel bereits intensiv beleuchtet. Nun zur Gegenwart. Heutzutage gehören Patterns zum Standardrepertoire im Software Engineering. Bei der Analyse existierender APIs wie Node.js, JDK, .NET, JQuery oder ganzer Softwaresysteme findet die Entwicklerin eine Vielzahl eingesetzter Muster.

Die Menge an Literatur zum Thema füllt gefüllt einen australischen Truck. Von nicht funktionalen Eigenschaften (Sicherheit, Zuverlässigkeit, Parallelität, Usability ...) über spezielle Domänen (Telekommunikation, Logistik, Embedded, Enterprise Architektur ...) bis hin zu konkreten Plattformen (Java, .NET, Ruby, Spring ...) lassen sich mittlerweile für fast jeden Zweck Entwurfsmuster ausfindig machen. Es gibt also mehr als die in „Design Patterns: Elements of Reusable Object-Oriented Design“ vorgestellten.

Abgesehen davon verhilft der Umgang mit Patterns zu einem systematischen Herangehen an Entwurfsprobleme. Auch dann übrigens, wenn diese nicht um Patterns kreisen.

Einige Wünsche wurden aber bisher nicht erfüllt. Exemplarisch sei das Thema Pattern-Sprachen genannt. Solche Sprachen existieren allenfalls für kleinere Domänen oder sind so mächtig, dass der Architekt sich bisweilen ohnmächtig fühlt. Dass sich dies in Zukunft gravierend ändern könnte, scheint unwahrscheinlich. Das Anwenden von Entwurfsmustern ist eine punktuelle Ange-

legenheit und nicht jede noch so kleine Problemlösung ist ein Pattern-Kandidat. Die Menge des Glue-Codes zwischen eingesetzten Mustern überwiegt also immer noch deutlich.

Neue Aufgabenbereiche für neue Patterns

Stiefmütterlich behandelte Themen wie Embedded Systems, Robotik und Internet of Things zeigen, dass es dort ein Potential für passende Patterns gibt. Bisher decken die meisten Muster eher Enterprise-Architekturen ab.

Vorstellbar wäre auch, dass ein zukünftiges Autorenteam Bücher wie das der GoF einer intensiveren Retrospektive unterzieht und die dokumentierten Pattern-Kataloge und -systeme gründlich refaktoriert. Immerhin haben die Entwurfsmuster bereits zwei Jahrzehnte auf dem Buckel. Vielleicht hält dann auch das funktionale Paradigma verstärkt Einzug in die Pattern-Welt. Eines jedoch steht fest: Patterns werden auch in Zukunft zum essentiellen Handwerkszeug in der Softwareentwicklung gehören. (ka)

Prof. Dr. Michael Stal

arbeitet als Principal Engineer in der Corporate Technology der SIEMENS AG und berät interne Kunden bei strategischen Produkten. Zudem hat er eine Professur am Software-Engineering-Lehrstuhl der Universität von Groningen inne.

Literatur

- [1] Christopher Alexander; A Timeless Way of Building; Oxford University Press, 1979
- [2] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides; Design Patterns; Elements of Reusable Object-Oriented Design; Addison-Wesley, 1995
- [3] Frank Buschmann, Hans Rohnert, Regine Meunier, Peter Sommerlad, Michael Stal; Pattern-Oriented Software Architecture; Volume 1: A System of Patterns; John Wiley & Sons, Inc., 1996
- [4] William J. Brown, Raphael C. Malveau, Hays W. McCormick, Thomas J. Mowbray; AntiPatterns; Refactoring Software, Architectures, and Projects in Crisis; John Wiley & Sons, Inc., 1998

Alle Links: www.ix.de/ix1608094

IMMER EINE IDEE SCHLAUER.



2 x Mac & i mit 30% Rabatt testen!

Ihre Vorteile:

- Plus: digital und bequem per App
- Plus: Online-Zugriff auf das Artikel-Archiv *
- Lieferung frei Haus

Für nur
13,80 €
statt 19,80 €

Jetzt bestellen und von den Vorteilen profitieren:

**mac-and-i.de/
minabo**

0541 80 009 120
leserservice@heise.de

* Für die Laufzeit des Angebotes.

+ Artikel-ARCHIV



Mac & i.
Das Apple-Magazin
von c't.