



h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi

FACHBEREICH INFORMATIK

Thema:

Das Architekturmuster

„Chaos zu Struktur (Mud-to-structure)“

Im Rahmen des Seminares:

Wissenschaftliches Arbeiten in der Informatik 1

Name, Vorname	Sterzel Kornelia
Datum	15.06.2016
Studiengang	Informatik
Betreuerin / Seminarleitung	Prof. Dr. Inge Schestag

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Stellen sind unter Angabe des Literaturverzeichnisses gekennzeichnet. Das gilt auch für Quellen, die ich selbst für andere Zwecke erstellt habe. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner Prüfung oder Prüfungsbehörde eingereicht worden.

Diese Arbeit weist 37.582 Zeichen (ohne Leerzeichen) auf, exklusive Abbildungen, Bildern, Graphiken etc.

Darmstadt, den 15.06.2016

Zusammenfassung

1. Architekturmustern und deren Entstehung

Die Idee für Architekturmuster stammt ursprünglich aus der Baubranche vom Architekt Christopher Alexander der mit Hilfe einer Sammlung an Entwurfsmustern, die zukünftigen Bewohner in den Entwurfsprozess mit einbinden wollte. Dieser Ansatz konnte sich allerdings nicht durchsetzen. Er diente allerdings als Vorlage für Kent Beck und Ward Cunningham 1987 die, die Idee aufgriffen und in die Softwarearchitektur übertrugen. Sie entwickelten Entwurfsmuster für die Erstellung von grafischen Benutzeroberflächen in Smalltalk.

2. Die Art wie und wofür sie angewendet werden

Architekturmuster sind hier probate Hilfsmittel um sich Anwendungsstrukturen von Softwaresystemen klarzumachen und diese übersichtlich und verständlich zu strukturieren. Dabei hilft das Muster nicht nur in den genauen Umfang der Anwendung und deren Schnittstellen nach außen zu ermitteln sondern auch um die im Projekt benötigte Ressourcen abzuschätzen. Um aus dem Informations- „Chaos“ eines Kundenwunsches etwas Kontrollierbares und auf Dauer stabiles System zu machen braucht es Strukturmuster (Mud-to-structure).

3. Der Einsatz von Architekturmustern im Projektablauf

Da in der Softwareentwicklung effizient Probleme gelöst werden sollen helfen uns ähnliche schon bekannte Problemstellungen beim Lösen neuer Aufgaben. Das erste Muster unter dem Begriff **Layers-Muster** aus diesem Themengebiet haben wir bereits kennengelernt mit dem *Open Systems Interconnection Model (OSI-Layers-Model)*. Ein weiteres Muster ist das **Pipes-and-Filters-Muster** mit dem wir in Betriebssysteme und Datenbanken arbeiten werden lernen folgt hier. Und ein drittes Muster, das **Blackboard-Muster** kommt aus dem Bereich der Spracherkennungssoftware. Diese Mustern und deren Einsatzgebiete möchte ich ihnen näher bringen.

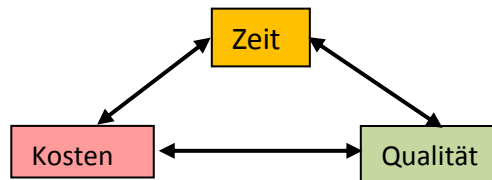
Inhaltsverzeichnis

1. Muster.....	4
1.1. Musterentstehung.....	4
1.2. Mustertypen.....	5
1.2.1. Architekturmuster	5
1.2.2. Entwicklungsmuster	6
1.2.3. Idiome.....	8
2. Architekturbeschreibungssprachen	9
2.1. UML.....	9
2.2. Diagramm Gruppierungen	9
2.3. Diagramme	9
3. Architekturmuster und Standards	11
2.4. Layers-Muster	13
2.4.1. Anwendung OSI.....	13
2.4.2. Anwendungsbeispiel	16
2.5. Pipes-und-Filters-Muster	16
2.6. Anwendung	17
2.7. Blackboard-Muster	17
2.8. Anwendung	18
3. Chaos zu Struktur (Mud-to-structure)“	19
3.1. Systementwurf.....	19
3.2. Muster finden	21
4. Zusammenfassung und Fazit	22

1. Muster

Unter Mustern in der Softwareentwicklung versteht man die Beschreibung einer häufig auftretenden Problemstellung. Sie liefern einen allgemeinen Lösungsansatz, da sie sprachneutral und plattformunabhängig formuliert sind. Beim Entwickeln eines Softwaresystems kommt es darauf an möglichst schnell, fehlerfrei, preisgünstig Ergebnisse zu liefern.

Hierfür steht das „Magische Dreieck“



Wenn ein Unternehmen ein Softwareprojekt plant, benötigt es einen möglichst genauen Plan über die Faktoren des magischen Dreiecks. Mit Hilfe von Projektplanung legt man das Budget und den Zeitplan fest. Abweichungen können zu Vertragsstrafen und sogar zu Projektabbruch führen.

Muster ermöglichen es, von den Erfahrungen der Spezialisten einen guten Lösungsansatz zu bekommen und somit von ihnen zu lernen. Dazu kommt, dass wenn die eigene Lösung gut ist, liegt sie häufig nahe bei dem entsprechenden Pattern. Pattern erhöhen die Wartbarkeit und sorgen für leicht nachvollziehbare Strukturen. Diese helfen zusätzlich noch bei der Projektaufteilung in Projektgruppen. Jede Gruppe kann dann ihren Teil entwickeln. Das Zusammenfügen geht über Schnittstellen von Komponenten einfacher, da die einzelnen Projektgruppen nur die geplante Schnittstelle zur Verfügung stellen müssen. Muster helfen zudem bei der Projektdokumentation und erhöhen mit ihrer Darstellbarkeit das Projektverständnis und die Geschwindigkeit beim Einarbeiten in dieses bei Personalwechsel. [JPA16]

Kaum zu glauben, dass diese Idee ursprünglich gar nichts mit Software zu tun hatte siehe 1.1

1.1. Musterentstehung

Denn die Idee für Lösungsmuster stammt ursprünglich aus der Baubranche vom Architekten Christopher Alexander, der mit Hilfe einer Sammlung an Entwurfsmustern die zukünftigen Bewohner in den Entwurfsprozess von Häusern einbinden wollte. Er definierte den Begriff des Musters so:

>>>Jedes Muster ist eine dreiteilige Regel, die eine Beziehung zwischen einem bestimmten Kontext, einem Problem und einer Lösung beschreibt. <<< [BL00 S. 247 ff]

Dies lässt sich auch so auf Softwareprojekte übertragen fand damals Kent Beck. Er und Ward Cunningham übertrugen 1987 das Konzept in die Softwarewelt. Sie entwickelten zuerst Entwurfsmuster für die Erstellung von grafischen Benutzeroberflächen in Smalltalk.

Bei grafischen Oberflächen kommt es zu häufig wechselnden Inhalten in der Darstellung, welche schnell zur Ansicht gebracht werden müssen. Dabei ist es ungünstig, wenn ständig auch die Funktionen angepasst werden müssen; daher entwickelten sie ein neues Lösungskonzept die Design Pattern.

Die Nutzung von Lösungsmustern konnte sich durchsetzen, da der Mensch gerne Probleme mit Hilfe von bekannten Lösungsmustern löst, wie man aus der Hirnforschung weiß. Muster gibt es in der Softwareentwicklung auf verschiedenen Ebenen siehe unter 1.2., wohingegen sich der Ansatz in der Baubranche nicht durchsetzen konnte. [BL00]

1.2. Mustertypen

Es gibt zwei große Gruppen von Mustern im Bereich der Softwareentwicklung. Diese sind auf einer hohen Ebene die Architekturmuster und auf der Ebene des Feinentwurfs die Entwicklungsmuster auch Design Pattern genannt. Die Begriffe werde ich im Weiteren noch erläutern unter 1.2.1 und 1.2.2.

Das wichtigste bei der Beschreibung von Mustern insgesamt ist ein sprechender *Mustername*, der sein *Einsatzgebiet* beschreibt und die Motivation, welche das entsprechende Problem schildert. Wenn dann noch die Struktur und die Art der Interaktion festgelegt sind, lässt sich mit seiner Hilfe eine Lösung entwickeln. Auch die Kommunikation im Projekt ist leichter, da die Muster eine allgemeine Konvention darstellen. Wichtig ist die Modularität vor allem bei Software, die lange genutzt und modifiziert werden soll, da dabei eventuell einzelne Module getauscht werden müssen, wofür auch Objekt-Orientierung nötig ist. [BL00] Da man die Muster zur leichteren Verbreitung und Nutzung bildlich beschreiben sollte, folgt unter Kapitel 2 eine kurze Einführung in Architekturbeschreibungssprachen, im Besonderen die Klassendiagramme der UML.

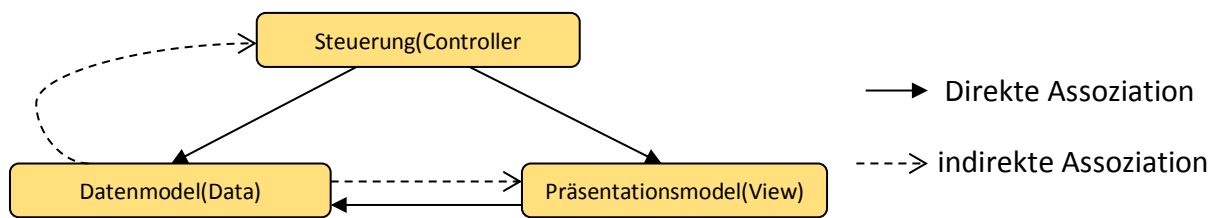
Diese Diagramme helfen neuen Entwicklern im Team die Einarbeitung ins Projekt; gerade das Muster in der Darstellung als Klassendiagramm erleichtert das Verständnis.

1.2.1. Architekturmuster

Architekturmuster (engl. *Architectural Pattern*) bilden die oberste Ebene der grundsätzlichen Struktur eines Softwaresystems ab. Der jeweilige Zuständigkeitsbereich qualifiziert das Architekturmuster für seine Aufgabe und initiiert deren Regeln und die nötigen Beziehungen. Sie sind Schablonen und werden auch Templates genannt und bilden eine Grundstruktur und die Beziehungen der Subsysteme oder Komponenten untereinander ab.

Mit Hilfe des Architekturmusters erhalten wir für eine bestimmte Problemstellung einer Softwarearchitektur eine Komponentenübersicht. Es bildet somit die Funktionalität der

Benutzeroberfläche ab z.B. mit Hilfe des **Model-View-Controller**-Musters. Was gerade auch bei großen Projekten die Modularität gewährleistet und bei der Kommunikation mit allen Projektbeteiligten hilft.



Das MVC trennt die Daten von der Ansicht und der Steuerung um z.B. verschiedene Views mit einer Steuerung bedienen zu können und bei Änderung der Daten die Anzeige nicht mit-ändern zu müssen. [BL00]

1.2.2. Entwicklungsmuster

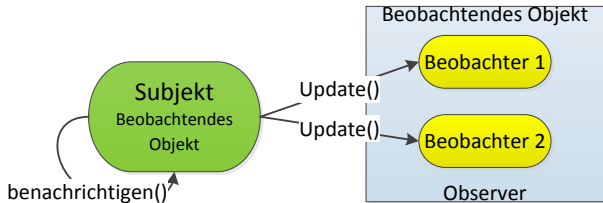
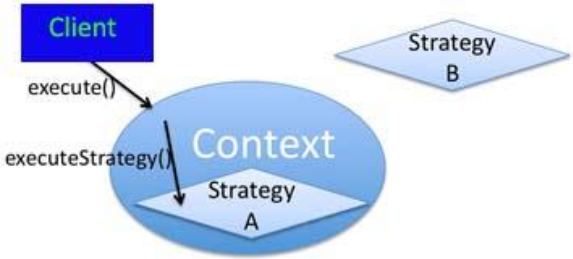
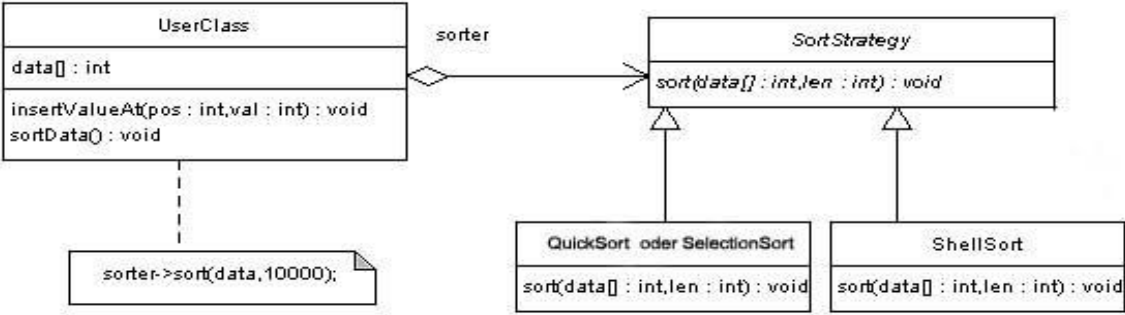
Die *Entwicklungsmuster* (engl. *Design Pattern*) bilden im Gegensatz zu Architekturmustern Subsysteme auf der Mittleren Ebene ab. Sie werden bei der Verfeinerung der Softwarestruktur der Architekturmuster um die Umsetzung des Projektes eingesetzt. Mit ihrer Hilfe kann man z.B. Abhängigkeiten mindern und wenn man das Observer Pattern als Beispiel betrachtet erhöht man die Performance in dem man Polling verhindert. Es entsteht dabei eine leichter erweiterbare und wartbare modulare Struktur. Auch die Entwicklungsmuster sind noch Programmiersprachen unabhängig; erst bei der Implementierung ist eine Programmiersprache nötig. Design Pattern sind Teil des Feinentwurfs, bei dem die Art der Umsetzung festgelegt wird.

Es gibt Design Muster für verschiedene Aufgabengebiete hier möchte ich nur eine kleine Auswahl nennen z.B. das Observer Pattern (Bildbeispiel unter 1.2.2. 1), welches eine Verfeinerung der Struktur des Model-View-Controller darstellt. Es hilft auf Ereignisse, bei denen sich der darzustellende Inhalt ändert zu reagieren. Das Strategie Pattern hilft die Erweiterbarkeit von Software zu erleichtern.

Dies macht man um Abhängigkeiten zwischen Komponenten zu minimieren, was gleichzeitig die Wartbarkeit verbessert. Allerdings kann die Verwendung von Mustern die Performance mindern, da mehr Klassen nötig sind um die Muster umzusetzen da Daten gekapselt werden, indem neue Verwaltungsklassen dazukommen.

Um die Vorteile einzelner Muster noch zu verbessern und zu ergänzen ist es möglich, Muster zu kombinieren. Es gibt allerdings auch Konstellationen die man auf keinen Fall tun sollte, sogenannte Antipattern bei denen man z.B. alle Funktionen in einer Klasse unterbringt. Der

Begriff für so eine zieh-alles-an-Klasse ist „Bombe“. Die Inhalte stehen ausführlicher in den Vorlesungsunterlagen der Vorlesung vom Fach Software Engineering von Prof. Akelbein SS2016. Da das Thema hier Architekturmuster lautet, belassen wir es hier bei der kurzen Vorstellung der schon genannten Designs bzw. Anti Pattern. [JPA16]

1.2.2. 1 Observer Pattern	1.2.2. 2 Strategie Pattern
<p>Das Subjekt enthält eine Liste der Beobachter und benachrichtigt sie bei Änderungen. Dazu kommt eine Methode an-/abmelden mit der sich die Beobachter registrieren können. Bei Änderungen werden alle Beobachter informiert. Diese können sich dann die Informationen abholen.</p>	<p>Ermöglicht große Flexibilität durch man Objekt und Verhalten unabhängig kapselt. Beim Strategie Pattern werden Klassen, die sich nur im Verhalten unterscheiden, getrennt von ihrem Verhalten Implementiert. Damit lassen sich Verhaltensweisen von unabhängigen Objekten benutzen, ohne jedes Mal wieder neu implementiert zu werden. Es mindert also Verhaltensredundanzen.</p>
	
<div data-bbox="188 1178 1374 1525"> <p style="text-align: right;">Observer</p> </div> <div data-bbox="188 1581 1318 1895">  </div>	
<p>Strategie Bildvorlagen alle aus dem Studienprojekt „Design Pattern“ von Herrn Philipp Hauer 2009-10</p>	

Hilfe beim Implementieren von Designpattern bieten Idiome. Sie stellen Mustercode zum Umsetzen in der gewählten Programmiersprache zur Verfügung. Der Mustercode ist allerdings noch nicht vollständig, da die Probleme zu vielfältig sind, um für jedes Problem eine gute und vollständige Lösung bereitzustellen.

Kommen wir nun zur eigentlichen Implementierung. Für die bekannten Muster gibt es in den gängigen Programmiersprachen wie C++, die man unter dem Begriff Idiome zusammenfasst. Ganz im Gegenteil zu den Mustern sind Idiome sprachenabhängig.

Hier findet man eine Erklärung wie man es umsetzt anhand von Beispielen für wichtige Design Muster:

<http://www.philippbauer.de/study/se/design-pattern.php>

geprüft 13.06.2016

1.2.3. Idiome

Die Idiome sind Implementierungen von Design Pattern in einer bestimmten Programmiersprache. Um zu zeigen wie so eine Implementierung aussehen kann; hier habe ich Codevorlagen in C++ eine Auswahl der wohl bekanntesten Design Pattern gefunden:

<https://www.cplusplus.net/forum/155350-full>

Es macht sicher Sinn, sich als Softwareentwickler mit noch nicht so großer Erfahrung die wichtigsten Muster anzusehen und sich zu merken, für was man sie gut einsetzen kann. Dies wird helfen sich in größere Projekte einzuarbeiten. Der eine Grund ist der Profit an Erfahrung und der andere ist der, mit den anderen Projektteilnehmern sich leichter austauschen und diskutieren zu können. Da es noch ein Script mit dem Thema Pattern gibt, soll hier nur verwiesen werden sich dort zu informieren und bei Interesse diese Unterlage durchzulesen.

2. Architekturbeschreibungssprachen

Architectur Description Languages (ADL) sind probate Hilfsmittel um gerade größere Softwareprojekte zu planen und zu dokumentieren. Mit Hilfe von einzelnen Diagrammen der *UML* lassen sich Ansichten für einzelne Entwicklungsgruppen und deren Spezifischen Anforderungen erstellen.

2.1. UML

Unified Modeling Language ist die wohl bekannteste Form der Modellierung für Softwareprojekte. Sie wurde von der **Object Management Group (OMG)** erstmals 1990 in Umlauf gebracht. Entwickelt wurde sie von Grady Booch, Ivar Jacobson und James Rumbaugh und ist sowohl von ihr als auch von der ISO (ISO/IEC 19505 für Version 2.4.1) standardisiert. Sie wurde immer wieder modifiziert und aktualisiert bis auf die heutige Version 2.5 (Juni 2015). [Wiki15]

Dieser 2.5er Standard erlaubt es auch Elemente anderer Diagramme zu verwenden um die Verständlichkeit zu erhöhen.

2.2. Diagramm Gruppierungen

Die UML Diagramme lassen sich in Strukturdiagramme, Verhaltensdiagramme und Interaktionsdiagramme unterteilen wie schon bekannt aus der UML.

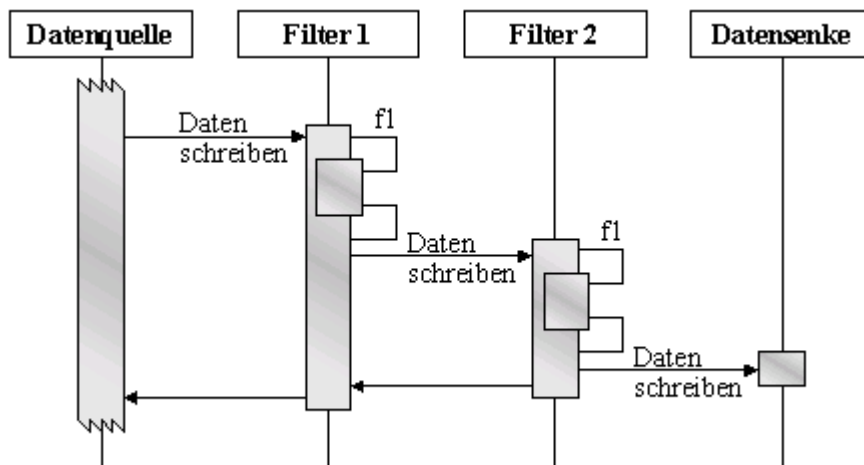
2.3. Diagramme

Die wichtigsten Diagramme im Bereich Architekturmuster sind das Klassendiagramm und das Sequenzdiagramm. Mit Hilfe des Klassendiagramms entsteht beim Modellieren die Übersicht über die Grundstruktur und die Schnittstellen modelliert und Methoden des Musters modelliert man mit dem Sequenzdiagramm. Am leichtesten macht man sich das Einsatzgebiet mit Hilfe von Use-Case-Diagrammen klar die nur die Benutzerinteraktion darstellen. Wenn man diese hat sollte man sich über die Subjekte wobei wir Näheres können sie in UML Büchern nachlesen. Hier möchte ich auf UML 2 kompakt verweisen, da man hier die Diagrammtypen schnell und übersichtlich findet.

Hier möchte ich als Beispiel für ein geeignetes Diagramm vorwegnehmen für das Pipes-and-Filters Muster denn es gibt kein Diagramm was dieses Muster übersichtlicher darstellt.

Mit dem Sequenzdiagramm kann man am besten Funktionsaufrufe darstellen und was diese tun. Da in Linux ruft jeder Funktionsaufruf in der Linux Shell einen Systemaufruf und die Erstellung eines neuen Prozesses dar. Wenn zwei Prozesse in unterschiedlichen Prozessräumen miteinander kommunizieren wollen benötigen sie Pipes welche man am

besten mit dem Sequenzdiagramm darstellt. Dieses Diagramm wird im Folgenden benötigt um Pipes-and-Filters besser zu verstehen.



Dieses Pipes and Filters Muster stammt von

<http://www.fh-wedel.de/~si/seminare/ws97/Ausarbeitung/3.Krutscher/archmu1.htm>

es bildet die Struktur dieses Musters und die waagrechten Rechtecke stellen deren Klassenobjekte dar.

3. Architekturmuster und Standards

Architekturmuster beschreiben Strukturierungsprinzipien zur Organisation von Softwaresystemen. Mit ihrer Hilfe teilt man das Softwaresystem in Subsysteme auf und legt deren Aufgaben fest. Die Aufgabengebiete bekommen eigene Schnittstellen um sicher zu stellen, dass sie, wenn nötig, modular sind. Das erleichtert spätere Erweiterungen.

Da wir uns an dieser Stelle noch auf hoher Abstraktionsebene befinden sind hier noch keine Festlegungen auf eine bestimmte Entwicklungsumgebung oder Programmiersprache wichtig. Es geht hier erst mal darum, sich grundlegende Strukturen klar zu machen. Diese Strukturierung hilft beim Festlegen der Kommunikationsschnittstellen der Komponenten eines Systems untereinander, diese so zu strukturieren, dass jeder Projektteilnehmer sich leichter einen Überblick verschaffen kann.

Um zu zeigen, wieso man einen solchen Dokumentier-Aufwand überhaupt betreibt, folgen die Architekturmuster mit Beispielen wo man sie bereits benutzt, aber sich oft nicht bewusst macht, dass es so ist. Jeder benutzt den Computer als Hardware mit Betriebssystem und Anwendungssoftware. Wenn man hier genauer hinsieht, findet man die Architekturmuster genau hier. Da man die Muster kennen muss um sie zu sehen, wenn sie einem begegnen, beginnen wir erst mit den Kategorien; und dann folgen die Musterbeschreibungen um schließlich im Kapitel 4 zu lernen, wie man sie findet.

Die Architekturmuster teilen sich in folgende Kategorien auf:

	Architekturmuster
Verteilte Systeme	Broker, Pipes-and-Filters, Microkernel
Interaktive Systeme	Model-View-Controller, Presentation-Abstraction-Control
Adaptierbare Systeme	Microkernel, Reflection
Mud to Structure	Layers, Pipes-and-Filters, Blackboard

Vorlage S.376 f [BL00]

Es gibt Architekturmuster Kategorien für die Einsatzgebiete in denen sie eingesetzt werden. Die Einteilung in Problemkategorien ist allerdings, wie anhand der obigen Tabellenauszug aus Pattern orientierter Softwarearchitektur schon ersichtlich ist, nicht so genau, denn es gibt

Überschneidungen in verschiedenen Bereichen gibt; dies liegt daran, dass es nicht so scharfe Trennungskanten gibt.

- **Verteilte Systeme:** Verteilung von Aufgaben auf verschiedene unabhängige Rechner, die auch unterschiedliche Betriebssysteme haben können. Man nutzt dies vor allem um die Kapazitäten zu vergrößern. Deren Bedeutung hat aber nachgelassen in Zeiten, in denen Hardware billiger und schneller geworden ist.
 - Das *Broker Muster* bildet die komplette Infrastruktur für verteilte Anwendungen ab. Bei Interesse findet man mehr darüber in den Standards der **Object Management Group**.
- **Interaktive Systeme:** Bilden den Bereich der Kommunikation zwischen Mensch und Maschine ab. Die Bedeutung von Benutzerinteraktion ist hingegen größer geworden durch Touchscreens bei Laptops und andere interaktive Systeme.
 - Das *Model-View-Controller Muster* trennt wie wir schon wissen, die Daten von der Darstellung und dem Controller um nicht ständig fragen müssen, ob sich was geändert hat stattdessen werden wir informiert bei einer Änderung und können uns die Daten holen.
 - Das *Presentation-Abstraction-Control* erweitert die Idee des MVC auf mehrere Hierarchien, wenn die Interaktionen mit unterschiedlichen Implementierungen umgesetzt werden müssen.
- **Adaptierbare Systeme:** Sie müssen schnell und in größerem Maß angepasst werden können. In der schnelllebigen Zeit, in der ständig neue Hardware auf den Markt gebraucht wird und auch die Software schneller upgedatet wird, ist Anpassbarkeit sehr wichtig geworden. Lernende Systeme müssen, wenn sie z.B. lernend sind und stabil sein sollen, müssen sie eine stabile Basis haben. Hierfür eignet sich der Microkernel.
 - Das *Microkernel-Muster* wird angewendet bei adaptierbaren Systemen auf einen kleinen Kern mit wenigen gekapselten Funktionalitäten und der Anwendungsteil wird ausgelagert, so dass wir keinen Plopp(Klasse die einen Großteil der Funktionalitäten und Attribute enthält).
 - Das *Reflection Pattern* wird eingesetzt wenn Objekte erst zur Laufzeit bekannt sind. Dieses Pattern ist mit Vorsicht zu verwenden, denn es kann nicht so einfach eine Fehlerbehandlung implementiert werden, ohne dabei das Objekt zu kennen.
- **Vom Chaos zur Struktur:** Ist ein Strukturierungsmuster der Architekturebene um ein System in Komponenten und Subsysteme aufzuteilen. Damit lässt es sich z.B. leichter

implementieren, da die Komponenten von einzelnen Entwicklergruppen erstellt werden können, da sie nur festgelegte Schnittstellen einhalten müssen aber die restliche Implementierung für andere Projektgruppen keine Rolle spielen. Dazu kommt, das verwendete Muster es neuen Projektteilnehmer erleichtern, sich ins Projekt einzuarbeiten. Modularität verbessert zudem Wartbarkeit, Erweiterbarkeit und die Stabilität.

- Layers: Werden auch Schichten genannt, ein Muster, das Systemanforderungen über verschiedene Ebenen abbildet. Weiter unter Punkt 3.1.
- Pipes-und-Filters: Ist ein Muster, das Daten über Pipes überträgt. Aufgaben werden auf die zugehörigen Ebenen aufgeteilt und bilden Filterkomponenten, die mit benachbarten Filtern kommunizieren. Filter lassen sich in diesem Muster beliebig anordnen, und je kleiner die Aufgaben der Filter, umso modularer sind sie. Mehr dazu unter Punkt 3.2.
- Blackboard: Es ist ein Muster für Probleme für die es noch keinen einfachen Lösungsansatz gibt. Dafür werden immer Näherungslösungen als Mitschrift und Entwicklungsgrundlage für die Lösung auf dem schwarzen Brett festgehalten, solange bis man den Lösungsbereich gefunden hat. Häufig findet man dieses Muster im Bereich Spracherkennungssoftware. Mehr unter Punkt 3.3. dazu.

2.4. Layers-Muster

Layers auch übersetzt mit Schichten, strukturiert Software in unterschiedliche Abstraktionsebenen. Diese Schichten kommunizieren untereinander und bieten einander Funktionen an. Wenn die Schichten auch zur Datenkapselung genutzt werden sollen und gleichzeitig Modularität gewährleistet werden soll, darf die Kommunikation nur von oben nach unten erfolgen.

2.4.1. Anwendung OSI

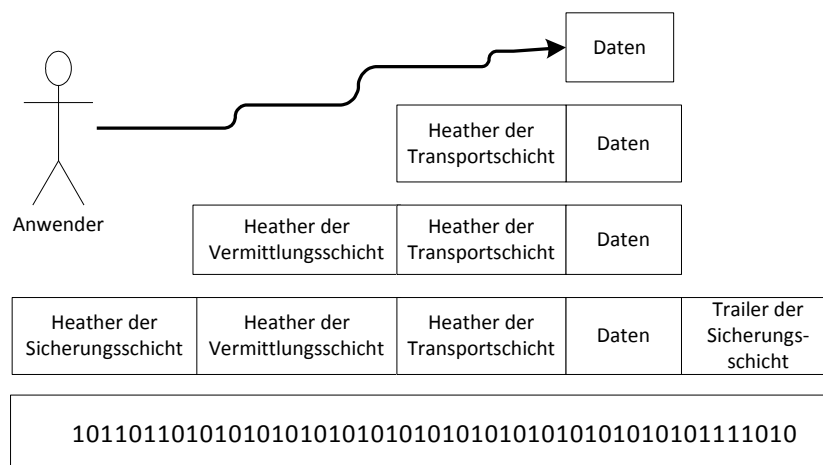
Das wohl bekannteste Anwendungsbeispiel stammt aus dem Bereich der Netzwerke mit dem heute stark genutzten Bereich der Webanwendungen. Ohne dieses Protokoll gäbe es wahrscheinlich keine Smartphones. Auch liegt es nah, dass auch die Anwendungsdichte von Laptops und Tablets nicht denkbar ist ohne die inzwischen unverzichtbare Möglichkeit des Datenaustausches über das World Wide Web.

Das OSI-Schichtenmodell (engl. *Open Systems Interconnection Model*) ist ein Layers-Muster, welches die Kommunikation auf einzelnen Ebenen abbildet. Es ist ein Referenzmodell für Netzwerkprotokolle, das Daten in Pakete zerlegt. Jede Schicht entnimmt entsprechend ihren Aufgaben Daten aus dem

Paket. Auch setzen hier Sicherheitsprotokolle an um zu prüfen, ob kein Datenverlust stattgefunden hat. Die Notwendigkeit dafür stammte ursprünglich aus instabilen Netzen. Heute brauchen wir sie stärker um festzustellen ob Daten auf dem Weg verändert wurden. Da Sicherheit eine immer größere Rolle spielt. Das OSI-Modell ermöglicht auch die Kommunikation über unterschiedlichste technische Systeme hinweg. Was beim Thema Verteilte Systeme und in der dazugehörigen Literatur beschrieben ist.

Für all diese Aufgaben entwickelte man Protokolle und Strukturierte sie in dem OSI sieben Schichtenmodell mit klar abgegrenzten Aufgaben für jede Schicht. Auch haben sie klare Schnittstellen und sind somit einfach austauschbar. Wie man anhand der folgenden Grafik sehen kann werden die von der jeweiligen Schicht benötigten Daten entnommen und der Rest weitergereicht.

Wenn ich jetzt die Transportschicht ändern muss, müssen die Anderen Schichten dies nicht wissen, solange die benutzten Schnittstellen gleich bleiben, also ihre Dienste gleich lassen. Dies habe ich in folgender Grafik mal veranschaulicht.



Jetzt könnte man sagen es handelt sich doch beim OSI Modell doch um ein 7-Schichten Modell. Die Grafik zeigt aber nur 4 Schichten. Das ist richtig, denn der Grund liegt darin, dass beim Internetprotokoll TCP/IP, was bei Mustern durchaus normal ist eine Implementierungs-Variation entstanden ist. Diese Umsetzung hat nicht alle Schichten auch umgesetzt, da sie wahrscheinlich einen Teil der Möglichkeiten als für die Aufgabe unnötigen Overhead ansah.

Bedenke bei der Umsetzung dass es gleichzeitig auch gilt Wartbarkeit, Entwicklungszeit und Kosten nicht aus dem Blick zu verlieren. Eben das Magische Dreieck vom Kapitel 1.

Um trotzdem alle Schichten aufzuführen, habe ich auf der nächsten Seite eine Tabelle der Schichten mit Verwendung und den entsprechenden Protokollen aufgestellt. Dies geschah in Anlehnung an die Vorlesung und das Script für Netzwerke von Herrn Baun. [CB12]

Nr.	Layer	Protokolle	Funktionen
7	Anwendungsschicht (engl. Application Layer)	HTTP, FTP, SMTP, POP3, DNS, SSH, Telnet	Ist die Schnittstelle zum Anwender auf der die Anwendersoftware wie z. B. Browser und Email Programme laufen. Protokolle welche mit Anwendungsprogrammen zusammen arbeiten.
6	Darstellungsschicht (engl. Presentation Layer)	Regeln zur Formatierung (Präsentation) von Nachrichten (heute veraltet)	Hier kann der Empfänger informiert werden, dass eine Nachricht z.B. im ASCII Format vorliegt, um deren Konvertierung zu ermöglichen.
5	Sitzungsschicht (engl. Session Layer) Synchronisation	Aufbau, Überwachung und Beenden einer Sitzung (Die Sitzungsschicht wird in der Praxis kaum angewendet, da alle Aufgaben dieser Schicht heute Anwendungsprotokolle übernehmen)	Ihre Aufgaben liegen im Aufbau, der Überwachung und dem Beenden der Sitzung. Die Sitzung ist eine virtuelle Verbindung zwischen zwei Anwendungen auf mindestens zwei physisch getrennten Rechnern. Die Dialogüberwachung, welcher Teilnehmer gerade spricht und auch die Regeln des Verbindungsauf- und -abbaus der Sitzungen legt man hier fest. Anhand von Kontrollpunkten kann bei Verbindungsabbruch zur Abbruchstelle zurückgekehrt werden und man muss nicht wieder von vorne beginnen.
4	Transportschicht (engl. Transport Layer) Segmente	TCP (Verbindungsorientierte Kommunikation), UDP (Verbindungslose Kommunikation)	Der Transport Layer ermöglicht den Transport zwischen den Prozessen in Segmenten(logische Adressierung). Die Transportschicht sorgt für die korrekte Datenübermittlung der Vermittlungsschicht an die entsprechenden Anwendungen. Sie teilt die Daten des Senders mit Transportprotokollen in kleinere Teile(Segmente) auf und leitet sie an das Protokoll der Vermittlungsschicht weiter. (TCP/IP ist heute Standard)
3	Vermittlungsschicht (engl. Network Layer) Paket	IP-Adressen (IPv4, IPv6), IPX	Der Network Layer versendet die Daten in Form von Paketen zwischen logischen Netzen und über den physischen Übertragungsabschnitt hinweg. Router sind Begrenzer logischer Netze. Meist über das verbindungslose IP-Protocol (der Vermittlungsweg wird nicht Protokolliert).
2	Sicherungsschicht (engl. Data Link Layer) Rahmen	MAC-Adressen, CSMA/CD, CSMA/CA Hardware: Bridges, Netzwerkkarten	Der Data Link ist für den Fehlerfreien Austausch von Daten zuständig. Die Sicherungsschicht ist zudem für die Fehlererkennung zuständig. Auch regeln die Protokolle des Data Link den Zugriff auf das Übertragungsmedium per CSMA/CD oder CSMA/CA. Die Sicherungsschicht begrenzt zudem die Bitfolgen in Rahmen sogenannten Frames. Sie markiert somit den Anfang und gewährleiste somit die gewünschte Zuverlässigkeit bei der Datenübertragung innerhalb eines physischen Netzes von einem Teilnehmer zum anderen. Der Rahmen enthält eine Prüfsumme anhand derer die Vollständigkeit überprüft wird und das Paket entweder verworfen oder korrigiert werden kann.
1	Bitübertragungsschicht (engl. Physical Layer) Bits	Ethernet, WLAN, ATM, FDDI, PPP, Token Ring	Der physical Layer überträgt Nullen und Einsen(Bits) und legt fest wie viele Bits pro Sekunde übertragen werden können. Zudem definiert sie ob die Verbindung in beide Richtungen gleichzeitig stattfinden kann.

2.4.2. Anwendungsbeispiel

So können sie Webseiten mit unterschiedlichen Browsern, auf unterschiedlichen Betriebssystemen anzeigen lassen. Wenn jemand sich z.B. die Homepage von Amazon anzeigen lässt, kann er das auf dem Smartphone, dem Laptop, dem Desktop oder auch dem Tablet tun. Da es unterschiedliche Geräte gibt haben wir auch unterschiedliche Systeme auf denen sie angezeigt werden will. Dazu kommt die unterschiedlichen Bildschirmgrößen und Browser-Implementierungen welche ebenfalls eine gleichaussehende Darstellung bieten.

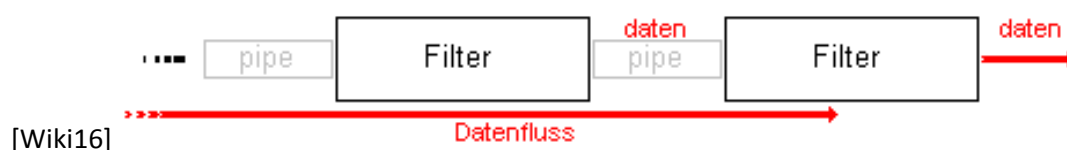
2.5. Pipes-und-Filters-Muster

Pipes und Filters Muster(engl. Pipes-and-Filters)stellt eine Struktur zur Verfügung, mit der man Datenströme verarbeitet. Dabei wird jeder Verarbeitungsschritt in einem eigenen Filter gekapselt. Der Filter kann ergänzend, verfeinernd oder transformierend wirken und dabei die Eingabedaten in eine andere Darstellungsform bringen. Die Datenübertragung zwischen zwei aufeinander folgenden Filtern geschieht über Kanäle(engl. Pipes). Die Filterfolge wird auch Verarbeitungspipeline genannt. Mit Hilfe von Pipes and Filters werden Daten inkrementell gelesen und geliefert, was die Auslastung verbessert, wodurch quasi parallele Verarbeitung ermöglicht wird.

Der Filter kann durch Eingabe Daten des Vorgänger Filters, durch Datenentnahme des Nachfolgers oder durch eine Schleife mit Entgegennahme von Daten und Ergebnisausgabe an den nächsten geschehen. Der passive Filter wird durch die Aktion eines anderen angestoßen, während der aktive selbstständig startet. Dies kann als Programm oder Thread geschehen. Verbindet man zwei aktive Filter müssen sie erst über den Kanal synchronisiert werden über einen First-in-First-Out Puffer. Passive Filter können auch aufgerufen werden, was aber die Variabilität de Filter erschwert.

Beispiele für eine Datenquelle wäre eine Datei oder ein Sensor, wodurch Daten desselben Typs geliefert werden, wobei sie aktiv dem ersten übergeben werden oder passiv einfach nur bereitgestellt werden. Der letzte Filter kann ebenfalls aktiv Ergebnisse entgegennehmen oder er muss dem Vorgänger erlauben, ihm die Daten reinzuschreiben. Das Sequenzdiagramm von S.9 bidet dieses Muster am klarsten ab. Dort sieht man wie die Schichten kommunizieren.

An dieser Stelle fragt man sich sicher, ob man dieses Muster benötigt, da die Struktur der Schichten Struktur ähnelt. Deshalb folgt ein Anwendungsbereich der belegt, dass es günstig ist.



2.6. Anwendung

Bei UNIX Systemen wird das Konzept erfolgreich beim Kommandointerpreter eingesetzt. Hier macht es das Betriebssystem performanter. Hier nutzt man die Flexibilität schnell Eingaben zu filtern und umzuwandeln mit einem Compiler. Die Eingabe-Abarbeitung ist bei der Konsole bekannt aus Betriebssysteme schnell und flexibel. Auch in dem Gebiet der Systemkommunikation bei verteilten Systemen finden wir diese Art von Kommunikation ebenfalls. Jede Kommunikation zwischen unterschiedlichen Prozessen, welche auf unterschiedlichen Speicherbereichen ihre Daten halten, kommuniziert mit Hilfe von Pipes. Wenn man sich vor Augen führt, dass man durch Änderung des Compilers beim Übersetzen eines C-Programmes nur mit Aufruf von g++ statt gcc ein ausführbare Datei erzeugt, die mit fast gleicher Eingabe erzeugt wurde, aber nur nach anderen Regeln liegt es nah das es sich um einen Filter handelt. Auch wenn man auf der Konsole eine Ausgabe macht oder die Daten in eine Datei umleitet, leistet dies ein Filter mit den Funktionalitäten wie vorher beschrieben.

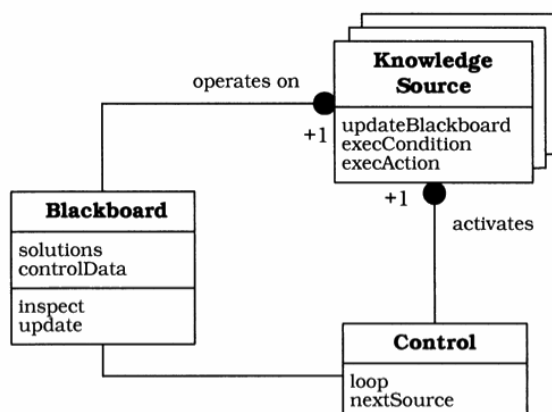
Vorteile: Filter neu anordnen ist leicht möglich.

Nachteile: gemeinsame Nutzung von Daten setzt voraus, das immer nur einer zugreifen kann und der andere, der auch noch an dieselben Daten will, warten muss bis der erste Benutzer fertig ist. Es kommt zum Prinzip der Queue, bei der jeder warten muss, bis er dran ist. Die Parallelverarbeitung ist dabei eher unwahrscheinlich. Das größte Manko ist allerdings die Schwierigkeit, eine Fehlerbehandlung zu machen, da diese hier nicht vorgesehen ist. Wenn eine Fehlerbehandlung nötig ist, sollte man besser auf das Schichten-Muster setzen. [SA16]

2.7. Blackboard-Muster

Das Blackboard stellt eine Datenstruktur dar, die beim Lösen von nicht *deterministischen* Problemen hilft. Da nicht Deterministische Probleme sich nicht einfach lösen lassen benötigt es oft eine Problemzerlegung in kleinere Teilprobleme die dann auch nur eine annäherungsweise Lösung ermöglichen.

Gefunden auf <http://wiki.hsr.ch/APF/Blackboard> am 14.6.2016



So sieht das entsprechende Muster aus, dass eine Tafel für die Notizen des Lösungsweges und Funktionen zum Aktualisieren zur Verfügung stellt.

2.8. Anwendung

Wenn man z.B. eine Software hat, welche Sprache erkennen soll gibt es an dieser Stelle keine hundertprozentig richtige oder falsche Lösung, stattdessen einen Näherungswert. Bedeutet, wenn ein Mensch ja sagt klingt, das je nach Tonlage anders oder auch wenn es anders betont wird, dennoch soll die Software das Wort richtig erkennen. Man suche also einen Bereich in dem man es als richtig wertet und dokumentiert dies. Somit gibt es für eine Sache nicht nur eine Lösungswert sondern mehrere. Genauso kommt es in der Wissenschaft immer wieder zu Bereichen in denen das Ergebnis nur im Näherungsbereich liegen muss, da exakte Werte nicht ausreichend häufig bei Wiederholung des Versuchs reproduzierbar sind.

3. Chaos zu Struktur (Mud-to-structure)“

Da wir die Muster jetzt schon kennen und wissen, dass sie dazu dienen Durcheinander von Komponenten und Objekten zu vermeiden und dabei zu helfen, zu einer Sinnvollen Zerlegung einer großen Aufgabe in zusammenspielende Teilaufgaben zu kommen. Geht es jetzt um das finden und benutzen, der Muster.

3.1. Systementwurf

Wenn wir einen neuen Systementwurf beginnen gilt es, sich die Anforderungen zu überlegen. Nehmen wir an, dass die gefundenen Spezifikationen vollständig und gut gewählt sind, muss dafür jetzt eine funktionale und stabile Struktur geschaffen werden, die auch Komponenten enthalten kann, welche nicht direkt zum Anwendungsgebiet gehören. Dazu kommt, dass auch Eigenschaften wie Portabilität, Wartbarkeit, Verständlichkeit und Stabilität die sehr wichtig sind.

Wenn man eine Software entwickelt, sollte man sich zuerst folgende Punkte überlegen:

- Was soll das System können?
 - Wer sind die Anwender / Anwendergruppen?
 - Wie sieht die Hard / Softwarebasis und die gegebenen Schnittstellen aus?
 - Was für einen Datenbestand gibt es und wie ist er verwaltet?
1. System: Ist die Aufgabe ein Betriebssystem, das die Hardware vor Schäden durch den Benutzer schützen muss, dann ist es sinnvoll, in Schichten zu modellieren. Dabei Hilft das Schichtenmodell, indem es der Anwenderschicht nur erlaubt, umliegende Schichten zu benutzen und nur mit deren Funktionen auch schützenswerte Bereiche zuzugreifen.

Auch setzt man dabei schon die erste Stufe der Objektorientierung und Modularität um, die auch leichtere Erweiterbarkeit gewährleistet.
 2. Jede Benutzergruppe möchte intuitiv, ohne großen Lernprozess interaktiv die Software bedienen können. Dies erreicht man dann z.B. mit dem MVC Muster, welches ermöglicht, verschiedene, oft ständig wechselnde Inhalte darzustellen. Hier macht es Sinn aus Performance Gründen die sich ändernden Daten melden zu lassen, dass neue Inhalte da sind und nicht das System mit warten auf Änderung zu blockieren.

Dies kennen wir von der Systemkategorie Interaktive Systeme, die entsprechende Architekturmuster kennt, wo man auch noch Vorschläge für mögliche Alternativmuster findet die unter Punkt 3 S.9ff auch noch etwas genauer beschrieben worden sind.

3. Schnittstellen benötigt man z.B. um die Kommunikation mit anderen Systemen, dem Internet, wie bei der Anwendung von Layers zu kommunizieren. Die Schnittstellen sind dabei festgelegt und alles drum herum ist beliebig. Die kann man sich klar machen daran, dass ich einen beliebigen Rechner mit meinem Wunsch Betriebssystem, wie Windows oder Linux, haben kann und mit Hilfe des Lieblingsbrowsers jede gewünschte Webseite ansehen kann. Auch das wechseln des Browsers ist kein Problem. Selbst mehrere gleichzeitig installiert zu haben ist möglich.

Um aber nicht nur dass zu gewährleisten, kombiniert man Muster auch noch häufig miteinander; denn auch die Kommunikation muss gesteuert werden und möglich sein. Hier wäre ein Beispiel für die Kommunikation über Pipes die Unix/Linux Konsole welche es ermöglicht mit Kommandos wie `ls -l` oder dem Kommando `Firefox` Programme aufzurufen und diese mit dem Betriebssystem und der Hardware kommunizieren zu lassen mit Hilfe von Pipes. Die Filter sorgen für die nötige Bearbeitung und Ausführung der Befehle mit Hilfe der Systemfunktionen der Shell. Wie im Script von Betriebssysteme von Professor Schütte [SA16] beschrieben.

Ein Nachteil hat allerdings die Verwendung von Pipes-and-Filters, denn sie kennt keine Ausnahme Behandlung. Wenn diese dringend nötig sein sollte, muss man über eine Kombination von Mustern nachdenken oder sich eine neue Lösung einfallen lassen, denn es gibt nicht für alles ein Muster.

4. Beim Datenbestand und der Verwaltung z.B. der Einbindung von vorhandenen Systemen sollte man den Bereich verteilte Systeme als Lösungshilfe in Betracht ziehen, wofür wir wissen, es die Muster Broker, Pipes-and-Filters, Microkernel gibt, welche eine Lösungshilfe sein können, so wie wir es schon aus der Liste von Systemgruppen und deren Architekturmustern auf Seite 9 kennen.

Was hier noch an Überlegung fehlt und häufig zu den Anforderungen der Kunden gehört ist die Realisierung von Vielseitigkeit wie z.B. bei Smartphones, was heute eine Immer größere Rolle spielt. Für die sogenannte Erweiterbarkeit von Systemen können die Muster der adaptierbaren Systeme helfen, wie schon erwähnt, aus der Liste Seite 9., deren Muster Microkernel, Reflection sind.

Wenn keines der Muster passt kann man sich überlegen ob sie zur Lösung modifizierbar sind in dem man sie kombiniert.

Das Blackboard Muster ist nicht so einfach zu finden, da es keine schon fertige Lösung für dieses Problem gibt, da es sich dabei um Lösungsbereiche handelt. Man findet es häufig bei Spracherkennungssoftware, da man verschiedene Stimmlagen und Betonungen desselben Wortes noch als dieses identifizieren muss, damit die Aufgabe erfüllt werden kann und die Software benutzbar ist.

3.2. Muster finden

Wenn man sich über die Grundsätzlichen Anforderungen einer Software Gedanken gemacht hat, kann man sich über Lösungsmöglichkeiten Gedanken machen.

1. Spezifizieren des Problems durch genaue Beschreibung des Problems im Kontext
2. Welche Regeln sind wichtig!
→ Stichworte: Fehlerbehandlung, Sicherheit, Performance
3. Gibt es vorgegebene Schnittstellen zu anderen Systemen
4. Muss es Viele verschiedene Screens darstellen können und ist es Interaktiv?
5. Muss es schnell und häufig modifiziert werden können?

Wenn dies geklärt ist soll man sich überlegen ob man schon ein Muster kennt, das passen könnte. Ist dies nicht der Fall, sucht man sich eine Passende Musterkategorie und schaut sich die enthaltenen Muster an. Passen sie gar nicht muss man eine komplett neue Lösung entwickeln. Sind nur Teile nicht so optimal kann man es vielleicht verbessern oder mit einem andren kombinieren so dass man die bestmögliche Näherung erzielt.

Es gilt immer nur Muster zu wählen, deren Vorteile passen und deren Nachteile nicht negativ ins Gewicht fallen.

Schließlich heißt es implementieren und Testen.

4. Zusammenfassung und Fazit

Muster helfen Entwicklern mit weniger Erfahrung von den Erfahrungen der Spezialisten zu profitieren. Zudem helfen sie dabei, sich über bekannte Strukturen ein Bild vom Gesamt System zu machen. Dies erleichtert die Einarbeitung in ein schon in Erstellung befindliches Softwareprojekt. Zusätzlich ist es auch Teil der Diskussionsgrundlage in der Projektentwicklung zwischen Entwicklern.

Man kann schließlich das Rad nicht ständig neu erfinden und schon erprobte stabile Lösungen verkürzen die Entwicklungszeit. Wenn man ein stabiles, wartbares auf viele Jahre erweiterbares System haben will, sollte man die Gelegenheit nutzen know-how und auch ganze Komponenten, welche im Unternehmen schon vorhanden sind wieder zu verwenden. Denn je mehr diese getestet sind (über die Jahre), umso sicherer kann man sich sein, dass sie keine großen Fehler mehr enthalten kann.

Zu 100% fehlerfreie Software weiß man, gibt es nicht. Die Annäherung an den Idealfall geschieht eben nur durch Testen, Testen und nochmal Testen. Einen 100%igen Testaufwand könnte gleichzeitig aber auch niemand leisten, da er unendlich viel Zeit und auch Geld verschlingen würde.

Um also aus dem Informations- „Chaos“ eines Kundenwunsches etwas Kontrollierbares und auf Dauer stabiles System zu machen, braucht es Strukturmuster (Mud-to-structure) und Design-Muster, welche den nötigen Aufwand für Stabilität und Fehlerbehebung minimieren. Das freut den Kunden und das Unternehmen.

Die Weiterentwicklung von Mustersystemen ist nicht statisch, denn es ist durchaus möglich das neue Muster gefunden werden und das alte verbessert oder verworfen werden. Also hat jeder eine Chance Wissen zu entwickeln und weiter zu geben.

Literaturverzeichnis

[BH05] Balzert, Heide (2005): UML 2 kompakt. Mit Checklisten. 2. Aufl. Heidelberg: Elsevier Spektrum Akad. Verl. (kompakt-Reihe).

[BL00] Buschmann, Frank; Löckenhoff, Christiane (2000): Pattern-orientierte Softwarearchitektur. Ein Pattern-System. 1., korr. Nachdr. München: Addison-Wesley.

[CB12] Dr. Christian Baun (SS2012): Netzwerke. Darmstadt.

[JM06] Jackson, Michael (2006): Software requirements & specifications. A lexicon of practice, principles and prejudices. [Nachdr.]. Harlow: Addison-Wesley (ACM Press books).

[PH10] Philipp Hauer (2009 - 2010): Das Observer Design Pattern. Leipzig (Studienprojekt). Online verfügbar unter <http://www.philippbauer.de/study/se/design-pattern/observer.php>, zuletzt aktualisiert am 2010, zuletzt geprüft am 14.06.2016.

[PH09] Philipp Hauer (2009 - 2010): Das Strategy Design Pattern. Leipzig (Studienprojekt). Online verfügbar unter <http://www.philippbauer.de/study/se/design-pattern/strategy.php>, zuletzt aktualisiert am 2010, zuletzt geprüft am 14.06.2016.

[PBG12] Posch, Torsten; Birken, Klaus; Gerdorf, Michael (2012): Basiswissen Softwarearchitektur. Verstehen, entwerfen, wiederverwenden. Heidelberg: dpunkt.verlag. Online verfügbar unter <http://gbv.eblib.com/patron/FullRecord.aspx?p=954656>.

[JPA16] Prof. Dr. Jens-Peter Akelein: Software Engineering. V1.3.1. Darmstadt.

[RH06] Reussner, Ralf; Hasselbring, Wilhelm (2006): Handbuch der Software-Architektur. 1. Aufl. Heidelberg: dpunkt.

[SH11] Starke, Gernot; Hruschka, Peter (2011): Software-Architektur kompakt. - angemessen und zielorientiert. 2., Auflage. Heidelberg Neckar: Spektrum Akademischer Verlag (IT kompakt).

[UC11] Ullenboom, Christian (2011): Java ist auch eine Insel. Das umfassende Handbuch ; [Programmieren mit der Java Platform, Standard Edition 6 ; Java von A bis Z: Einführung, Praxis, Referenz ; von Klassen und Objekten zu Datenstrukturen und Algorithmen]. 9., aktualisierte Aufl., 1., korrigierter Nachdr. Bonn: Galileo Press (Galileo computing).

[ZG10] Zöller-Greer, Peter (2010): Software-Architekturen. Grundlagen und Anwendungen ; mit einer Einführung in Architekturbeschreibungssprachen (ADLs), UML, Object-Z, OCL, CORBA, IDL, Entwurfsmuster, Architektursichten, Architekturmuster, DDD, Architektur-Dokumentation, Komplexitätsprobleme, Standard-Architekturen, SOA, TOGAF, RM-ODP, Software-Factories. 3. Aufl. Wächtersbach: Composita-Verl. (Reihe Wissen & Praxis "kompakt").

[Wiki16] Wikipedia: Architekturmuster. Online verfügbar unter <https://de.wikipedia.org/wiki/Architekturmuster>.

[SA16] Schütte (2016): Betriebssysteme. Prozesse. h_da, Darmstadt. Online verfügbar unter <https://www.fbi.h-da.de/~a.schuette/Vorlesungen/Betriebssysteme/>, zuletzt geprüft am 14.06.2016.

[Wiki15] Wikipedia (2015): UML. Online verfügbar unter https://de.wikipedia.org/wiki/Unified_Modeling_Language, zuletzt aktualisiert am 09.2015, zuletzt geprüft am 14.06.16.