Physikalische Chemie, Universität des Saarlandes

Vorlesung: Programmierkonzepte in den Naturwissenschaften im Sommersemester 2021

PD Dr. Till Biskup

— Glossar zu Lektion 17: „Entwurfsmuster“ —

Hinweis: Die nachfolgend genannten Begriffe und Definitionen erheben keinen Anspruch auf formale Korrektheit, sondern dienen lediglich dem besseren Verständnis der in der Vorlesung behandelten Themen und sind im jeweiligen Kontext zu sehen. Mehrfache, voneinander abweichende Definitionen in unterschiedlichen Kontexten sind daher möglich. Englische Begriffe werden zwar nach Möglichkeit übersetzt, erscheinen aber ggf. unter ihrem englischen Namen in der Liste. Verweise untereinander sind durch ↑ gekennzeichnet.

Abstraktion Nach Edsger Dijkstra das einzige mentale Werkzeug, das es erlaubt, eine große Vielzahl von Fällen abzudecken. Zweck der Abstraktion ist es nicht, vage zu sein, sondern im Gegenteil ein neues Bedeutungsniveau zu schaffen, das präzise Beschreibungen erlaubt.

Abstraktionsebene Summe aller ↑Abstraktionen eines bestimmten Abstraktionsgrades. ↑Funktionen (bzw. ↑Methoden) sollten immer nur Anweisungen enthalten, die zur gleichen Abstraktionsebene gehören.

Aggregation *aggregation*, „klassische“ Form der ↑Zusammensetzung in der ↑objektorientierten Programmierung, definiert eine klare „*hat ein*“-Beziehung

Assoziation *association*, Interaktion von ↑Objekten/↑Klassen auf die Weise, dass ein Objekt/eine Klasse einen Service für ein anderes Objekt/eine andere Klasse bereitstellt.

Attribut im Kontext der ↑objektorientierten Programmierung eine Variable, die innerhalb einer ↑Klasse definiert wird. ↑Methoden operieren auf den Attributen einer ↑Klasse bzw. dem daraus erzeugten ↑Objekt.

BDUF „*big design up front*“, Ansatz in der Softwareentwicklung, den kompletten Entwurf einer Software in großem Detail vor dem Beginn der Implementierung zu erstellen. Eine Alternative ist die iterative Entwicklung von unten nach oben (*bottom-up approach*), wie von der agilen Softwareentwicklung propagiert.

Design Pattern ↑Entwurfsmuster

Entwurfsmuster *design pattern*, Beschreibungen miteinander kommunizierender ↑Objekte und ↑Klassen, die maßgeschneidert sind, um ein generelles Entwurfsproblem in einem bestimmten Kontext zu lösen. Entscheidend für das Verständnis des Begriffs ist seine doppelte Natur. Entwurfsmuster sind sowohl die Beschreibung einer (wiederkehrenden) Problemstellung als auch der Strategie für die Lösung des jeweiligen Problems. Die ↑Abstraktionsebene von Entwurfsmustern liegt oberhalb jener von ↑Klassen und ↑Objekten als der grundlegenden Elemente der ↑objektorientierten Programmierung, aber gleichzeitig unterhalb der Gesamtarchitektur eines Systems (↑Softwarearchitektur). ↑Muster im weiteren Sinn finden sich auch auf weiteren ↑Abstraktionsebenen von Software.

Funktion im Kontext der ↑strukturierten Programmierung eine Liste von Anweisungen, die eine bestimmte Aufgabe erfüllt und der Programmiersprache unter einem festen Namen bekannt ist.

GoF *Gang of Four*, die Autoren des Werkes „*Design Patterns. Elements of Reusable Object-Oriented Software*“ (Addison-Wesley, Boston 1995), namentlich Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. Das Erscheinen dieses Buches gilt als eigentliche Geburtsstunde der ↑Entwurfsmuster in der Softwareentwicklung.

Hammer-Nagel-Problem Metapher für das Phänomen, dass gerade anfangs eine neu erlernte Strategie exzessiv und deshalb zu viel eingesetzt wird. Lässt sich nur durch Erfahrung lösen.

Heuristik die Kunst, mit begrenztem Wissen und wenig Zeit dennoch zu wahrscheinlichen Aussagen oder praktikablen Lösungen zu kommen; analytisches Vorgehen, bei dem mit begrenztem Wissen über ein System mit Hilfe von mutmaßenden Schlussfolgerungen Aussagen über das System getroffen werden.

Idiom Linguistik: Spracheigentümlichkeit; Softwaretechnik: Umsetzung (Implementierung) abstrakter ↑Muster bzw. Lösung einfacher Aufgaben (auf niedrigster ↑Abstraktionsebene) in einer konkreten Programmiersprache

Kapselung *encapsulation*, ein ↑Objekt enthält Daten (↑Attribute) *und* zugehöriges Verhalten (↑Methoden) und kann beides nach Belieben vor anderen Objekten verstecken.

Klasse *class*, im Kontext der ↑objektorientierten Programmierung die Blaupause für die Erzeugung eines ↑Objektes; Definition der Daten (↑Attribute) und des zugehörigen Verhaltens (↑Methoden).

Kopplung *coupling*, in Software der Grad der Verbindung zweier Komponenten. Programmierkonzepte zielen generell auf eine lose Kopplung einzelner Komponenten ab, da so die Wiederverwendbarkeit erleichtert wird.

Kristallkugel Nur in der Theorie funktionierendes Hilfsmittel für den Blick in die Zukunft, das u.a. hilfreich wäre, um Software bereits in ihrer Entstehung auf künftige Anforderungen hin auszulegen. Aufgrund anderer damit einhergehender Probleme ist die reale Funktionalität einer Kristallkugel nicht wünschenswert.

Methode im Kontext der ↑objektorientierten Programmierung eine ↑Funktion, die innerhalb einer ↑Klasse definiert wird und auf den ↑Attributen einer ↑Klasse bzw. dem daraus erzeugten ↑Objekt operiert.

Modularisierung Aufteilung der Gesamtaufgabe in kleinere Abschnitte. Die Aufteilung wird so lange fortgesetzt, bis die Lösung für den aktuellen Abschnitt unmittelbar in Form von Quellcode offensichtlich ist. Setzt die Definition von ↑Schnittstellen voraus.

Muster *pattern*, nach Christopher Alexander abstrakte Beschreibung eines wiederkehrenden Problems sowie einer generellen Lösung für dieses Problem, deren konkrete Ausgestaltung meist hochgradig individuell ist. In der Softwareentwicklung tauchen Muster auf unterschiedlichen Ebenen auf, angefangen bei ↑Idiomen über ↑Entwurfsmuster bis zu Mustern auf der Ebene der ↑Softwarearchitektur.

Objekt *object*, im Kontext der ↑objektorientierten Programmierung der grundlegende Baustein eines Programms, bestehend aus den Daten (↑Attribute) und dem zugehörigen Verhalten (↑Methoden).

objektorientierte Programmierung (OOP) ein ↑Programmierparadigma, bei dem Daten (Variablen zugewiesene Werte, als ↑Attribute bezeichnet) und Funktionen (↑Methoden), die auf diesen Daten (Attributen) operieren, eine Einheit bilden. Die in den ↑Attributen gespeicherten Daten lassen sich i.d.R. nur vermittelt durch (öffentlich zugängliche) ↑Methoden der ↑Klasse bzw. des daraus erzeugten ↑Objektes ansprechen. Es gibt eine klare Trennung zwischen öffentlicher ↑Schnittstelle und internen Verarbeitungsroutinen. Wichtige Vertreter objektorientierter Programmiersprachen sind Smalltalk, C++ und Java, aber auch Python.

Paradigma nach Thomas S. Kuhn ein Satz allgemein anerkannter wissenschaftlicher Leistun-

gen, der für eine gewisse Zeit einer Gemeinschaft von Fachleuten maßgebende Probleme und Lösungen liefert

Polymorphie *polymorphism*, „Vielgestaltigkeit", ähnliche ↑Objekte können auf die gleiche Botschaft in unterschiedlicher Weise reagieren.

Portierbarkeit Möglichkeit, eine Software bzw. einen Entwurf auf eine andere Architektur (Hardware) oder Programmiersprache (Software) zu übertragen. Eine zentrale Eigenschaft von Programmiersprachen ist die ↑Abstraktion der zugrundeliegenden Hardware, so dass derselbe Quellcode auf unterschiedlicher Hardware lauffähig ist. Abstraktere Konzepte wie ↑Entwurfsmuster setzen oft grundlegende Aspekte wie ↑objektorientierte Programmierung voraus, versuchen aber sonst, möglichst unabhängig von der jeweiligen Programmiersprache zu sein. ↑Idiome sind im Gegensatz dazu sprachspezifisch und nicht zwischen Programmiersprachen übertragbar.

Programmierparadigma ein ↑Paradigma der Art zu programmieren. Wichtige Beispiele sind ↑strukturierte Programmierung, ↑objektorientierte Programmierung und funktionale Programmierung.

Refactoring Verbesserung der Qualität des Quellcodes einer Software ohne Einfluss auf ihr von außen erkennbares Verhalten. Diszipliniertes Vorgehen zum Aufräumen von Quellcode, das die Wahrscheinlichkeit, Fehler einzuführen, minimiert.

Refactoring to Patterns Ansatz, durch ↑Refactoring eine gegebene Software auf die Verwendung von ↑Entwurfsmustern hin umzustrukturieren, ohne dabei ihr von außen erkennbares Verhalten zu verändern. Regelfall für die Anwendung von ↑Entwurfsmustern. Gleichzeitig der Titel eines Buches von Joshua Kerievsky (Addison-Wesley, Boston 2005).

Schnittstelle *interface*, Begriff mit mehreren leicht unterschiedlichen Bedeutungen; (1.) ↑Signatur einer ↑Methode. (2.) Im weiteren Sinne die Gesamtheit der öffentlichen ↑Attribute und ↑Methoden einer ↑Klasse bzw. eines ↑Objekts. Der Nutzer kennt nur die Schnittstelle, die Implementierung ist irrelevant und kann sich problemlos jederzeit ändern, solange die Funktionalität erhalten bleibt. Das dient der Trennung von Verantwortlichkeiten und ermöglicht ↑Modularisierung und ist in der Folge ein wesentlicher Aspekt der Softwarearchitektur.

Softwarearchitektur Aufteilung eines größeren Projektes in einzelne kleinere Projekte bzw. Aufgaben (↑Modularisierung), Definition klarer ↑Schnittstellen und Anforderungen sowie der Interaktion der einzelnen Teile miteinander.

strukturierte Programmierung ein ↑Programmierparadigma, das die Zahl möglicher Kontrollstrukturen auf nur zwei (Iteration, Selektion) beschränkt, insbesondere den `goto`-Befehl eliminiert (E. Dijkstra). Idealerweise hat ein Codeblock nur jeweils genau einen Ein- und Ausgang. Nach D. Knuth der systematische Einsatz von Abstraktion, der es ermöglicht, große Programme aus kleine(re)n Komponenten zusammenzusetzen. Wichtige frühe Vertreter strukturierter Programmiersprachen sind C und Pascal. Die meisten heutigen Programmiersprachen (mit Ausnahme der funktionalen Programmiersprachen) unterstützen die strukturierte Programmierung.

Subklasse ↑Klasse, die von einer anderen Klasse (der ↑Superklasse) ↑Attribute und ↑Methoden erbt. Die Vererbung geht dabei i.d.R. über die nach außen hin sichtbare ↑Schnittstelle der Superklasse hinaus. Die Subklasse erbt von der ↑Superklasse häufig nur den „kleinsten gemeinsamen Nenner" und implementiert die spezifische Funktionalität.

Superklasse ↑Klasse, von der andere Klassen (↑Subklassen) ↑Attribute und ↑Methoden erben. Die Vererbung geht dabei i.d.R. über die nach außen hin sichtbare ↑Schnittstelle der Superklasse hinaus. Superklassen implementieren bzw. definieren normalerweise nur das Notwendigste, sozusagen den „kleinsten gemeinsamen Nenner". Alle spezifische Funktionalität wird in der ↑Subklasse implementiert.

transzendieren die Grenzen eines Bereichs überschreiten

UML *Unified Modeling Language*, „vereinheitlichte Modellierungssprache“, grafische Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von Software-Teilen und anderen Systemen. UML definiert Bezeichner für die meisten bei einer Modellierung wichtigen Begriffe und legt mögliche Beziehungen zwischen diesen Begriffen fest. Darüber hinaus definiert sie grafische Notationen für diese Begriffe und für Modelle statischer Strukturen und dynamischer Abläufe, die man mit diesen Begriffen formulieren kann. Weite Verbreitung findet eine sehr kleine Untermenge der in UML vorhandenen Begriffe und grafischen Notationen für die grafische Darstellung der Beziehungen von ↑Objekten untereinander in der ↑objektorientierten Programmierung.

Vererbung (*inheritance*) Weitergabe aller Eigenschaften (↑Attribute, ↑Methoden) von einer ↑Superklasse an eine ↑Subklasse. Die Subklasse ist vom gleichen Typ wie die Superklasse, was wiederum die Grundlage der ↑Polymorphie ist. Änderungen der Superklasse wirken sich allerdings auf die Subklasse aus, die ↑Kapselung wird entsprechend geschwächt. Vgl. ↑Zusammensetzung.

YAGNI „*You ain't gonna need it*“, (nicht nur) in der angelsächsischen Programmierwelt verbreitetes Akronym und wichtige Regel für die Programmierung. Zielt darauf ab, jeweils nur das zu implementieren, was im Augenblick wichtig ist oder von dem zweifellos klar ist, dass es in Kürze gebraucht wird. Versuch, durch einen pragmatischen Ansatz ein „zu viel“ an Abstraktion zu vermeiden. Das zugrundeliegende Problem, das damit angegangen werden soll: Prognosen (hier: bzgl. der zukünftigen Anforderungen an eine bestimmte Software) sind schwierig, insbesondere wenn sie die Zukunft betreffen (vgl. ↑Kristallkugel).

Zusammensetzung *composition*, Wechselwirkung zwischen *unabhängigen* ↑Objekten, die ↑Kapselung bleibt im Gegensatz zur ↑Vererbung voll erhalten. Ein Objekt ist aus anderen Objekten zusammengesetzt. Die Objekte können anderweitig vollkommen unabhängig sein. Vgl. ↑Aggregation und ↑Assoziation.