Physikalische Chemie, Universität Rostock

Vorlesung: Wissenschaftliche Softwareentwicklung

Wintersemester 2023/24

Dr. habil. Till Biskup

— Glossar zu Lektion 18: „Refactoring“ —

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 [1] 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.

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.

Code Review Gemeinsame Begutachtung eines Quellcode-Abschnitts eines Projektes durch mehrere Projektbeteiligte. Ein Code Review ist formaler als das ↑Pair Programming, außerdem wird beim Code Review in der Regel *nicht* programmiert, sondern nur über den Quellcode diskutiert. Ziele eines Code Reviews sind u.a.: Wissensvermittlung innerhalb des Teams, Sicherstellen eines gemeinsamen Kenntnisstands, Entwicklung von Ideen für die Weiterentwicklung. Eine Verbesserung des diskutierten Codes geht meist damit einher, allerdings sollte Kritik immer konstruktiv sein.

DRY „*Don't repeat yourself*“, (nicht nur) in der angelsächsischen Programmierwelt verbreitetes Akronym und wichtige Regel für die Programmierung. Doppelungen im Code sind meist ein guter Hinweis darauf, dass eine ↑Abstraktion fehlt bzw. nicht erkannt wurde. Darüber hinaus sind Doppelungen im Code ein zentraler Grund für schlechte Qualität und Wartbarkeit.

Entwurfsmuster *design pattern*, Beschreibungen miteinander kommunizierender ↑Objekte und ↑Klassen, die maßgeschneidert sind, um ein generelles Entwurfsproblem in einem bestimmten Kontext zu lösen.[2, S. 3] 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 Elemen-

te 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.

intellektuelle Beherrschbarkeit *intellectual manageability*, nach Edsger Dijkstra das Hauptziel der Softwaretechnik (*software engineering*) – und letztlich des Projektmanagements. Unterschiedliche Lösungsansätze für ein Problem sind unterschiedlich gut intellektuell beherrschbar. Entsprechend ist die intellektuelle Beherrschbarkeit das zentrale Kriterium für die Entscheidung, welche Lösung für ein Problem bevorzugt wird.

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

Kohäsion innerer Zusammenhalt; hier: Zusammenhang einzelner Aspekte einer Softwareeinheit zueinander. Ein Ziel der Softwareentwicklung ist starke Kohäsion (*strong cohesion*). Jede Einheit (z.B. ↑Methode, ↑Funktion, ↑Klasse) hat eine Aufgabe, und alle Teile dieser Einheit dienen dem Zweck, diese eine Aufgabe zu erfüllen.

Kopplung hier: enge Bindung mehrerer Einheiten einer Software aneinander, so dass sie nicht unabhängig wiederverwendbar (bzw. ggf. auch nicht testbar) sind. Ein Ziel der Softwareentwicklung ist die lose Kopplung (*loose coupling*) der einzelnen Einheiten.

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.

Nebenwirkung *side effect*, Wirkung; in der theoretischen Informatik die Veränderung des Programmzustands einer abstrakten Maschine. In der Praxis der Programmierung meist die Auswirkung einer Zuweisung eines Wertes zu einer Variablen, die außerhalb des konkret betrachteten Kontextes liegt.

nichtorthogonales Design Das Gegenteil ↑orthogonalen Designs. Tritt häufig in der Realität auf und erschwert das Verständnis und insbesondere die Erweiterung und Veränderung von Code.

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.

orthogonales Design größtmögliche Unabhängigkeit einzelner Komponenten einer Software voneinander. Der Begriff kommt ursprünglich aus der Mathematik und lässt sich elegant an zwei orthogonalen Vektoren verdeutlichen: Bewegung parallel zu einem Vektor verändert nicht die Projektion entlang des anderen Vektors. Orthogonales Design in Software führt zu kompaktem Design selbst komplexer Zusammenhänge. Wichtige Aspekte sind starke ↑Kohäsion und geringe ↑Kopplung. Rein orthogonales Design ist frei von ↑Nebenwirkungen: Jede Operation verändert nur genau eine Sache ohne Einfluss auf andere Aspekte. Darüber hinaus gibt es jeweils nur genau einen Weg, eine Eigenschaft eines Systems zu verändern. [3, S. 89ff.], [4, S. 34ff.]

Pair Programming „Paarprogrammierung", enge und aktive Zusammenarbeit zweier Programmierer bei der Programmierung von Software. Eine Person sitzt an der Tastatur und programmiert, während die andere aktiv über den entstehenden Code und das Problem nachdenkt und ggf. korrigierend eingreift. Auffallende Probleme werden sofort angesprochen und diskutiert. Die Rollen sollten häufig wechseln (mindestens innerhalb einer Stunde), idealerweise ebenso die Zusammensetzung der Paare.

Paradigma nach Thomas S. Kuhn [5] ein Satz allgemein anerkannter wissenschaftlicher Leistungen, der für eine gewisse Zeit einer Gemeinschaft von Fachleuten maßgebende Probleme und Lösungen liefert

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. Setzt zwingend ausreichende (automatisierte) ↑Tests, idealerweise ↑Unittests, voraus.

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.

Test hier: strukturiertes Vorgehen, eine Software zu überprüfen. Setzt die Definition klarer Anfangs- und Endbedingungen (Eingabe und Ergebnis) voraus und sollte idealerweise vollständig automatisiert ablaufen können. Vgl. ↑Unittest.

Unittest ↑Test eines Codeblocks in Isolation. Ein Unittest überprüft von außen, ohne den Quellcode des zu testenden Systems zu kennen oder zu benötigen. Die getesteten Codeblöcke sind i.d.R. klein. Zwingende Vorausset-

zung ist, dass das (erwünschte) Verhalten des zu testenden Codeblocks eindeutig definierbar (und seinerseits in Form von Quellcode formalisierbar) ist. Unittests sind gewissermaßen die unterste Ebene (automatisierter) ↑Tests.

Literatur

[1] Edsger W. Dijkstra. The humble programmer. *Communications of the ACM* 15 (1972), S. 859–865.

[2] Erich Gamma u. a. *Design Patterns. Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley, 1995.

[3] Eric S. Raymond. *The Art of UNIX Programming*. Boston: Addison Wesley, 2004.

[4] Andrew Hunt und David Thomas. *The Pragmatic Programmer*. Boston: Addison-Wesley, 1999.

[5] Thomas S. Kuhn. *Die Struktur wissenschaftlicher Revolutionen*. Frankfurt am Main: Suhrkamp, 1976.