Physikalisch-Technische Bundesanstalt, Berlin (Adlershof)

Vorlesung: Wissenschaftliche Softwareentwicklung

2023/24

Dr. habil. Till Biskup

— Glossar zu Lektion 17: „Testautomatisierung und TDD“ —

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.

Äquivalenz hier: gleicher Wert bzw. Inhalt zweier Variablen bzw. Objekte (im weiteren Sinn). Beim Vergleich auf Gleichzeit zweier Objekte im weiteren Sinn ist relevant, ob die ↑Identität (dasselbe Objekt) oder die Äquivalenz (der gleiche Wert bzw. Inhalt) überprüft werden sollen.

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.

Boolescher Wert „Wahrheitswert“, einer der zwei Werte der Booleschen Algebra, repräsentiert durch „0“ („*False*“) und „1“ („*True*“).

Bug (engl. für „Wanze, Käfer“) Programmfehler oder Softwarefehler, allgemein ein Fehlverhalten von Computerprogrammen.

Clean Code „sauberer Code“, letztlich lesbarer Code, der insbesondere im Kontext der naturwissenschaftlichen Datenauswertung die essentiellen Kriterien von Wiederverwendbarkeit, Zuverlässigkeit und Überprüfbarkeit erfüllt.

Debugger Werkzeug zur Diagnose und zum Auffinden von Fehlern (↑Bug) in Programmen. Debugger bringen eine Reihe hilfreicher Funktionalitäten zur Analyse eines in einem Programmablauf aufgetretenen Fehlers mit und sind oft in einer IDE integriert.

Debugging Behebung von Fehlern (↑Bugs) in Software, häufig unter Verwendung eines ↑Debuggers. Sollte immer ein systematisches Vorgehen sein.

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 Entwurfsmus-

tern 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.

Framework Rahmenstruktur, Ordnungsrahmen; in der Softwaretechnik ein Programmiergerüst, das den Rahmen zur Verfügung, innerhalb dessen der Programmierer eine Anwendung erstellt. Die Struktur der individuellen Anwendung wird u.a. durch die im Framework verwendeten ↑Entwurfsmuster beeinflusst.

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

Identität hier: das identische Objekt (im weiteren Sinn). Beim Vergleich auf Gleichzeit zweier Objekte im weiteren Sinn ist relevant, ob die Identität (dasselbe Objekt) oder die ↑Äquivalenz (der gleiche Wert bzw. Inhalt) überprüft werden sollen.

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.

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 [3] 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.

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.

Signatur hier: Name und Parameter einer ↑Funktion bzw. ↑Methode, also alles, was ein Nutzer braucht, um diese Funktion oder Methode verwenden zu können.

Softwarearchitektur Aufteilung eines größeren Projektes in einzelne kleinere Projekte bzw. Aufgaben (↑Modularisierung), Definition klarer ↑Schnittstellen, Anforderungen und 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, [4]). Idealerweise hat ein Codeblock nur jeweils genau einen Ein- und Ausgang. Nach D. Knuth [5, S. x] 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.

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.

Test-Framework ↑Framework, das Funktionalität für das Schreiben von ↑Tests bereitstellt. Weit verbreitet sind die ↑xUnit-Frameworks.

testgetriebene Entwicklung *test-driven development*, Strategie der Softwareentwicklung, zunächst einen Test (i.d.R. ↑Unittest) zu schreiben und dann erst den Produktivcode. Vom Produktivcode wird nach Möglichkeit auch immer nur genau soviel geschrieben, um den initial fehlschlagenden Test erfolgreich zu durchlaufen. Das beugt der Implementierung letztlich nicht benötigter Funktionalität vor (↑YAGNI). Außerdem wird zunächst i.d.R. die einfachste denkbare Fassung des Produktivcodes implementiert, ohne sich um die Kriterien sauberen Codes (↑Clean Code) Gedanken zu machen. Deshalb ist der dritte wesentliche Schritt der testgetriebenen Entwicklung das ↑Refactoring.

Triangulierung auch Triangulation, in der Geodäsie das Aufteilen einer Fläche in Dreiecke und deren Ausmessung; hier: Verfahren der ↑testgetriebenen Entwicklung, durch mehrere ↑Unittests aus unterschiedlichen Richtungen (d.h. mit unterschiedlichen Parametern) die zu testende Einheit zu überprüfen und so schrittweise zur gewünschten Funktionalität zu kommen.

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 Voraussetzung 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.

xUnit-Framework ↑Test-Framework, das in Smalltalk entwickelt wurde, über Java (JUnit) weite Verbreitung fand und heute für viele Programmiersprachen existiert. Stellt sehr viele Funktionen und ↑Muster, die sich insbesondere für das Schreiben von ↑Unittests bewährt haben, zur Verfügung.

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).

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

[4] Edsger W. Dijkstra. Go to statement considered harmful. *Communications of the ACM* 11 (1968), S. 147–148.

[5] Donald E. Knuth. *Literate Programming*. Stanford: Center for the Study of Language and Information, 1992.