Buch: Softwareentwicklung für die Naturwissenschaften

Dr. habil. Till Biskup

— Glossar zu Kapitel 13: „Namen“ —

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.

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.

CamelCase Schreibweise, bei der zusammengesetzte Worte direkt miteinander verbunden werden und jeder Wortteil mit einem Großbuchstaben beginnt. I.d.R. werden Abkürzungen, die in Großbuchstaben erscheinen, komplett groß geschrieben und das nächste Wort mit einem Großbuchstaben begonnen, z.B. `HTTPResponse`.

Editor Programm zum Erstellen von Quellcode, der dann entweder kompiliert oder interpretiert und ausgeführt werden kann. Grundsätzlich ist für (fast) alle Programmiersprachen ein reiner Texteditor ausreichend. Oftmals steigern integrierte Entwicklungsumgebungen (↑IDE) die Produktivität allerdings ganz erheblich.

Eineindeutigkeit Bijektivität, Eindeutigkeit in beiden Richtungen. Mathematisch die Abbildung eines Elements einer Menge auf genau ein Element einer zweiten Menge und umgekehrt, weshalb Definitions- und Zielmenge die gleiche Mächtigkeit aufweisen. Im Kontext von Namen bedeutet das: Jeder Name bildet auf *genau ein* Konzept ab, so dass man eindeutig vom Namen auf das dahinterstehende Konzept und vom Konzept eindeutig auf den Namen schließen kann.

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

funktionale Programmierung ein ↑Programmierparadigma, das sich stark an der mathematischen Definition von Funktionen orientiert und im strengen Sinn auf die explizite Zuweisung von Werten zu Variablen verzichtet. Wichtige Elemente sind Listen und die ↑Rekursion. Durch den Verzicht auf explizite Zuweisungen sind funktionale Programme frei von Seiteneffekten (↑Nebenwirkung) und eignen sich damit hervorragend für Parallelisierung. Wichtige Vertreter funktionaler Programmiersprachen sind Lisp, Haskell und Erlang, die Konzepte finden sich mittlerweile aber auch in vielen anderen Programmiersprachen, darunter Python, C++ und Java.

IDE integrierte Entwicklungsumgebung, engl. *integrated development environment*; Software zur Programmierung (Quellcode-Erstellung), die neben einem ↑Editor als Hauptkomponente noch diverse weitere Werkzeuge integriert. Häufig integriert sind der Zugriff auf die Versionsverwaltung, die Build-Umgebung und ein Debugger sowie Werkzeuge zur statischen Codeanalyse. Darüber hinaus unterstützt die ↑Editor-Komponente einer IDE

nicht nur (komplexe) Syntax-Hervorhebung, sondern auch Aspekte wie ↑Refactoring. Nachteile einer IDE sind die steile Lernkurve aufgrund der erheblichen Komplexität dieser Programme. Eine gute IDE und ihre souveräne Beherrschung durch einen Programmierer haben andererseits erheblichen Einfluss auf Qualität und Geschwindigkeit des Programmierens.

Iteration eine von zwei Kontrollstrukturen der ↑strukturierten Programmierung, neben der ↑Selektion. Meist als Schleife implementiert, die über alle Elemente einer Liste läuft und für jedes Element bestimmte Anweisungen ausführt.

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

Konvention innerhalb einer Gruppe oder einem (lokalen) Kontext getroffene (temporäre) Festlegung. Ziel von Konventionen ist die Vereinheitlichung und damit einhergehend die Befreiung von der Notwendigkeit, jedesmal aufs Neue nachdenken zu müssen, wie z.B. gewisse Prozesse durchgeführt oder Objekte benannt werden sollen. Konventionen sind im Gegensatz zu ↑Standards weniger verbindlich und deutlich flexibler sowie *ad hoc* innerhalb einer Gruppe einführbar.

Lösungsraum (*solution domain*) Kontext der Programmierer eines Programms, Gegensatz zum ↑Problemraum. Namen aus dem Lösungsraum bestehen i.d.R. aus Begriffen aus der Welt der Programmierung.

magische Zahl i.d.R. jede Zahl außer „0“ und „1“, die im Quellcode vorkommt. Sollte *immer* einer Variablen bzw. Konstanten mit sprechendem Namen zugewiesen werden und dann nur noch indirekt über diese Variable/Konstante verwendet werden.

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.

Namensraum (*namespace*) Kontext, innerhalb dessen der Name eines Objektes (im allgemeinen Sinn, also Variable, Funktion, Methode, Objekt, Klasse, ...) eindeutig ist. Wird häufig durch Funktionen/Methoden oder in der objektorientierten Programmierung durch Objekte/Klassen hergestellt. Der gleiche Name kann in unterschiedlichen Namensräumen zur Bezeichnung unterschiedlicher Objekte (im allgemeinen Sinn) verwendet werden. Namensräume dienen damit der Organisation und Strukturierung.

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. Da in der ↑funktionalen Programmierung auf Zuweisungen gänzlich verzichtet wird, sind funktionale Programmiersprachen frei von Nebenwirkungen, was sie für die parallele Verarbeitung prädestiniert.

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 [1] ein Satz allgemein anerkannter wissenschaftlicher Leistungen, der für eine gewisse Zeit einer Gemeinschaft von Fachleuten maßgebende Probleme und Lösungen liefert

Problemraum (*problem domain*) Kontext der Fragestellung, die mit einem Programm (d.h. Software) angegangen werden soll, Gegensatz zum ↑Lösungsraum. Namen aus dem Problemraum verweisen i.d.R. auf Konzepte, mit denen die Anwender eines Programms vertraut sind (aber nicht notwendigerweise die Programmierer/Entwickler).

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.

Rekursion Aufruf einer Funktion in ihrer eigenen Definition. Allgemein lässt sich eine Rekursion immer auch als ↑Iteration und umgekehrt ausdrücken. Da die ↑funktionale Programmierung das Konzept der ↑Iteration nicht kennt, ist hier nur die Rekursion möglich. Die Definition über eine Rekursion ist meist eleganter, die über eine ↑Iteration oft effizienter, was den Speicherbedarf angeht.

strukturierte Programmierung ein ↑Programmierparadigma, das die Zahl möglicher Kontrollstrukturen auf nur zwei (↑Iteration, ↑Selektion) beschränkt, insbesondere den `goto`-Befehl eliminiert (E. Dijkstra, [2]). Idealerweise hat ein Codeblock nur jeweils genau einen Ein- und Ausgang. Nach D. Knuth [3, 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.

Selektion eine von zwei Kontrollstrukturen der ↑strukturierten Programmierung, neben der ↑Iteration. In den meisten Sprachen über Bedingungen (`if...else...end`) realisiert, die sich ggf. beliebig verschachteln lassen.

Standard von einem oft internationalen und anerkannten Gremium definierte Festlegung. Standards sind im Gegensatz zu ↑Konventionen sehr viel starrer und nicht *ad hoc* von einer Gruppe einführbar.

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. ↑dynamische Codenanalyse und ↑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 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.

Literatur

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

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

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