Buch: Softwareentwicklung für die Naturwissenschaften

Dr. habil. Till Biskup

— Glossar zu Kapitel 16: „Dokumentation im Code“ —

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.

Anwenderdokumentation Art der Dokumentation, die sich in erster Linie an die Nutzer eines Programms wendet und deshalb deutlich weniger softwaretechnisch als eine ↑Entwicklerdokumentation formuliert ist, dafür aber meist von einer Kenntnis der für die Software relevanten Problemstellung ausgeht. Im Fokus steht im Gegensatz zur ↑Entwicklerdokumentation gerade *nicht* die Implementierung der Funktionalität, sondern ihre Nutzung. Typische Aspekte einer Anwenderdokumentation sind ein kurzer Überblick über die Merkmale eines Programms, einfache (einführende) Beispiele, konkrete Anwendungsszenarien und ggf. ein strukturiertes und umfangreiches Nutzerhandbuch. Eine gute Anwenderdokumentation ist mit erheblichem Aufwand verbunden, sorgt aber für eine deutlich bessere Nutzbarkeit (und damit Verbreitung) eines Programms.

API *application programming interface*, Programmierschnittstelle oder genauer Schnittstelle zur Anwendungsprogrammierung

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.

Aufrufgraph *call graph*, grafische Darstellung des Kontrollflusses in einer Software, also der Aufrufe von Subroutinen (d.h. ↑Funktionen oder ↑Methoden) untereinander. Dadurch wird die Beziehung der unterschiedlichen Subroutinen untereinander offensichtlich, und zyklische Abhängigkeiten durch wechselseitigen Aufruf (ggf. über mehrere Zwischenschritte) lassen sich erkennen.

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

Coding Conventions Innerhalb der Entwicklergruppe eines Projektes zumindest zu einem gegebenen Zeitpunkt festgelegte Konventionen zu bestimmten Aspekten der Formatierung von Quellcode. Dient der Vereinheitlichung und trägt dadurch wesentlich zur Lesbarkeit bei. Wichtiger als der Inhalt ist die konsequente Befolgung der Konventionen und der möglichst breite Konsens über die Inhalte innerhalb der Entwicklergruppe.

Compiler Im Deutschen meist als „Übersetzer“ bezeichnetes Programm, das den Quellcode eines Programms in direkt auf der Hardware ausführbaren Maschinencode übersetzt. Kompilierte Sprachen sind im Gegensatz zu interpretierten Sprachen (↑Interpreter) meist deutlich schneller, aber in binärer Form (Maschinencode) an eine spezifische Hardwareplattform gebunden.

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.

Entwicklerdokumentation Art der Dokumentation, die sich in erster Linie an Entwickler, d.h. Programmierer, wendet. Wesentlicher Bestandteil einer solchen Dokumentation ist i.d.R. die ↑API-Dokumentation, die sich meist automatisch aus den Kommentarköpfen im Quellcode generieren lässt. Auch ↑Coding Conventions und Ideen für die weitere Planung sind oft Teil einer solchen Entwicklerdokumentation, sowie die Beschreibung derjenigen Alternativen, die betrachtet aber nicht gewählt wurden. Siehe auch: ↑Anwenderdokumentation

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

IDE integrierte Entwicklungsumgebung, engl. *integrated development environment*; Software zur Programmierung (Quellcode-Erstellung), die neben einem ↑Editor als Hauptkomponente noch diverse weitere Werkzeuge integriert. 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.

Interpreter Im Gegensatz zum ↑Compiler ein Programm, das den Quellcode eines Programms nicht in direkt ausführbaren Maschinencode übersetzt, sondern den Quellcode einliest, analysiert und ausführt. Die Übersetzung des Quellcodes erfolgt entsprechend zur Laufzeit des Programmes, was die Ausführungsgeschwindigkeit gegenüber kompilierten Programmen (↑Compiler) in der Regel deutlich reduziert. ↑Skriptsprachen werden in der Regel interpretiert und nur selten kompiliert.

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.

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.

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 inter-

nen 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

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

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.

selbstdokumentierend mehrere Bedeutungen, (1) Eigenschaft von Auswertungssoftware, alle Prozessierungsschritte (automatisch) zu dokumentieren. Wesentliche Voraussetzung für die Nachvollziehbarkeit wissenschaftlicher Datenverarbeitung und -Auswertung und damit für deren Wissenschaftlichkeit. (2) Quellcode, der u.a. durch geschickte Wahl der Namen von Variablen, Funktionen, Methoden, Objekten, Klassen, ... weitgehend ohne Kommentare lesbar und nachvollziehbar ist. Letztlich der „heilige Gral“ der Programmierung und Ziel sauberen Quellcodes (↑Clean Code).

selbstdokumentierender Code Code, der ↑selbstdokumentierend ist. Im gegebenen Kontext in der zweiten Bedeutung von ↑selbstdokumentierend verstanden.

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

Skriptsprache Meist vergleichsweise einfach zu erlernende Programmiersprache, die i.d.R. interpretiert (↑Interpreter) und nur selten kompiliert (↑Compiler) wird. Die Unterschiede zwischen Skriptsprache und interpretierter Programmiersprache sind fließend. Beispiele sind: Python, Perl, PHP, bash.

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.

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.

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.