Buch: Softwareentwicklung für die Naturwissenschaften

Dr. habil. Till Biskup

— Glossar zu Kapitel 28: „Datenverarbeitung und -Analyse“ —

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.

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 die Nutzung einer Anwendung bzw. eines Programms. 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.

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

Dokumentation im Kontext eines ↑Systems zur Datenverarbeitung mehrere Aspekte, angefangen von der ↑Anwenderdokumentation und ↑Entwicklerdokumentation verwendeter Software bis zum Protokoll aller Verarbeitungsschritte von Daten eines Datensatzes (↑Historie). Vgl. ↑selbstdokumentierend (1.).

Domain-Driven Design Ein gutes ↑Modell der komplexen Realität und Fragestellung bildet die Grundlage für den ↑Kern einer Anwendung.

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 Versionsnummern die Eindeutigkeit der Zuordnung einer Versionsnummer zu *genau einem* Zustand der Software und umgekehrt der Zuordnung eines Zustandes der Software zu *genau einer* Versionsnummer. 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.

Entwicklerdokumentation Art der Dokumentation, die sich in erster Linie an Entwickler, d.h. Programmierer, wendet und auf die Weiterentwicklung der Software fokussiert. Wesentlicher Bestandteil einer solchen Dokumentation ist i.d.R. die ↑API-Dokumentation, die sich meist automatisch aus den Kommentarköpfen im Quellcode generieren lässt. Die Aspekte einer externen Projektdokumentation (Anforderungsanalyse, High-Level-Design, Roadmap, Changelog, 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. Bei einer Bibliothek ist die Entwicklerdokumentation relevant für die Nutzung derselben zur Entwicklung darauf basierender eigener Programme. Vgl. ↑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. Vgl. ↑Methode.

größeres Projekt hier: Alles, was mehr als zwei Wochen Arbeit kostet und deutlich mehr als zweihundert Zeilen (reinen) Quellcode bzw. mehr als eine Handvoll Unterfunktionen umfasst. Wichtig ist der Fokus: Sobald ein Programm über längere Zeit und/oder von anderen verwendet werden soll (was eher die Regel statt die Ausnahme ist), ist es ein größeres Projekt.

Historie im Kontext eines ↑Systems zur Datenverarbeitung ein (vollständiges) Protokoll aller Prozessierungsschritte auf den Daten eines Datensatzes. Das Protokoll sollte neben *allen* Parametern eines Prozessierungsschrittes auch die Versionsnummer der jeweils verwendeten Routine enthalten, die in Verbindung mit einer ↑Versionsverwaltung erlaubt, genau die jeweils verwendete Fassung (↑Revision) wiederherzustellen bzw. zu untersuchen. Notwendige Voraussetzung für ↑reproduzierbare Wissenschaft.

Infrastruktur Personelle, sachliche und finanzielle Ausstattung, um ein angestrebtes Ziel zu erreichen. Im Kontext der Softwareentwicklung die Gesamtheit der Hilfsmittel, die (manche) Abläufe formalisieren und für Struktur und Überprüfbarkeit sorgen. Erleichtert die Arbeit des Programmierers, indem sie viele Aspekte festlegt, die so zur Routine werden (und keine Denkleistung absorbieren).

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

Kern der Anwendung Implementation des zugrundeliegenden ↑Modells in Software, d.h. Abbildung auf Code, zumeist in Form abstrakter ↑Klassen. Vgl. ↑Domain-Driven Design.

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

Komplexität Eigenschaft der Realität, dass selbst sehr wenige einfache Regeln in ihrer Kombination ein nichttriviales Verhalten erzeugen können, das den Menschen für gewöhnlich überfordert. ↑Größere Projekte in der Softwareentwicklung sind immer komplex, ein ↑System zur Datenverarbeitung allemal. Eine zentrale Strategie zum Umgang mit Komplexität ist ↑Abstraktion und in der Folge ↑Modularisierung. Nach Fred Brooks [2] lassen sich zwei Arten von Komplexität unterscheiden: ↑vermeidbare Komplexität und ↑unvermeidliche Komplexität.

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.

Mehrfachvererbung *multiple inheritance*, eine ↑Klasse erbt (↑Vererbung) von mehr als einer ↑Superklasse. Wird von den wenigsten Programmiersprachen unterstützt, oftmals behilft man sich hier aber des Konzeptes einer ↑Schnittstelle (*interface*) (3.) und kann dann mehr als eine solche implementieren (bzw. davon erben). Konzeptionell lassen sich diese beiden Ansätze quasi identisch einsetzen.

Metadaten Informationen zu den numerischen Daten, notwendige Voraussetzung für eine sinnvolle Verarbeitung der Daten im Kontext eines ↑Systems zur Datenverarbeitung und für ↑reproduzierbare Wissenschaft.

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.

Modell sprachliche Formulierung des ↑Problemraums und seiner entscheidenden Zusammenhänge und Abläufe auf hohem Abstraktionsniveau; Summe der „Geschäftsregeln“

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.

monolithisch aus einem Stück bestehend; zusammenhängend und fugenlos

Objekt *object*, im Kontext der ↑objektorientierten Programmierung der grundlegende Baustein eines Programms, bestehend aus den Daten (↑Attribute) und dem zugehörigen Verhalten (↑Methoden). Ein Objekt ist in diesem Kontext immer die Instanz einer ↑Klasse.

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

Polymorphie *polymorphism*, „Vielgestaltigkeit“, ähnliche ↑Objekte können auf die gleiche Botschaft (den Aufruf einer gleichnamigen ↑Methode) in unterschiedlicher Weise reagieren.

Prinzip der geringsten Überraschung *principle of least surprise*, Regel für die Programmentwicklung, soweit möglich auf in einem gegebenen Kontext etablierte Regeln und Standards zurückzugreifen. Baustein für möglichst intuitive Bedienbarkeit.

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 und ↑Abstraktionen, 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.

proprietär auf herstellerspezifischen, (meist) nicht veröffentlichten Standards basierend

Repository Versionsdatenbank, (zentraler) Speicherort der versionierten Dateien im Kontext einer ↑Versionsverwaltung.

Repräsentation Darstellung von Charakteristika von Daten entweder grafisch oder tabellarisch. Repräsentationen sollten sich, wann immer möglich, automatisch aus den zugrundeliegenden Daten erzeugen lassen. In diesem Fall ist es nur notwendig, die Vorschrift zur Erzeugung abzulegen, nicht die Repräsentation selbst. Das ermöglicht außerdem einfache Änderungen der Darstellung.

reproduzierbare Wissenschaft *reproducible science*, seit der Etablierung rechnergestützter Datenauswertung eigentlich nie mehr erreichter, aber für die Wissenschaft konstituierender Aspekt, dass sich Ergebnisse und Auswertungen unabhängig reproduzieren lassen, weil alle dazu notwendigen Aspekte vollständig und ausreichend beschrieben wurden. Motivation für die Vorlesung, deren Ziel es ist, die Hörer mit Konzepten vertraut zu machen, die letztlich eine ernstzunehmende reproduzierbare Wissenschaft ermöglichen.

Revision einzelner der ↑Versionsverwaltung bekannter Zustand eines ↑Repositorys

Schnittstelle *interface*, Begriff mit mehreren leicht unterschiedlichen Bedeutungen; (1.) ↑Signatur einer ↑Funktion oder ↑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. (3.) In einer weiteren Bedeutung wird der Begriff (auch im Deutschen dann häufig mit seinem englischen Pendant) für (abstrakte) Klassen verwendet, die lediglich eine Schnittstelle (im Sinne von 2.) definieren. Das ist hauptsächlich dann von Bedeutung, wenn die Programmiersprache keine ↑Mehrfachvererbung unterstützt, aber das Implementieren von „*Interfaces*“.

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 und in der Folge ↑reproduzierbarer Wissenschaft. (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).

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 und Anforderungen sowie der Interaktion der einzelnen Teile miteinander. Nach Robert C. Martin die Gestalt eines Systems, die ihm von seinen Entwicklern gegeben wird: Unterteilung des Systems in Komponenten, ihre Anordnung, und die Art ihrer Interaktion miteinander. [4, S. 136]

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.

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 „kleins-

ten 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.

System zur Datenverarbeitung hier: Gesamtsystem für wissenschaftliche Datenverarbeitung von der Datenaufnahme bis zur fertigen Publikation, das alle Aspekte umfasst und das ↑reproduzierbare Wissenschaft möglich macht und gewährleistet. Definitiv ein ↑größeres Projekt, das nicht nur eine ↑monolithische Anwendung umfasst, sondern viele Aspekte darüber hinaus. Setzt entsprechende ↑Infrastruktur und in der Umsetzung der einzelnen Komponenten sauberen Code (↑Clean Code) und eine solide ↑Softwarearchitektur voraus.

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.

Typisierung *typing*, Zuweisung eines Typs zu einem Objekt (im abstrakten Sinne) einer Programmiersprache, z.B. Ganzzahl (*integer*) oder Zeichenkette (*string*) im Fall einer Variable. ↑Abstraktion, die die Ausdrucksstärke von Programmiersprachen und Programmen deutlich erhöht, und die Überprüfung der Korrektheit erleichtert sowie Optimierungen ermöglicht. Typisierung kann explizit und implizit erfolgen. Darüber hinaus wird zwischen starker und schwacher Typisierung sowie zwischen statischer und dynamischer Typisierung unterschieden. Jede Art der Typisierung hat ihre Vor- und Nachteile, und unterschiedliche Programmiersprachen verwenden unterschiedliche Arten der Typisierung.

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.

unvermeidliche Komplexität *essential complexity*, nach Fred Brooks [2] jener Teil der ↑Komplexität eines Systems, der in der Komplexität der Fragestellung (↑Problemraum) begründet ist und der sich nicht verkleinern lässt. Eine gute ↑Softwarearchitektur zielt auf die Beherrschung dieser unvermeidlichen Komplexität u.a. durch Einsatz von ↑Abstraktion und ↑Modularisierung. Vgl. ↑Domain-Driven Design, ↑vermeidbare Komplexität.

Vererbung *inheritance*, Weitergabe aller Eigenschaften (↑Attribute, ↑Methoden) von einer ↑Superklasse an eine ↑Subklasse. Die Subklasse ist vom gleichen Typ (↑Typisierung) 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.

vermeidbare Komplexität *accidential complexity*, nach Fred Brooks [2] jener Teil der ↑Komplexität eines Systems, der *nicht* in der Komplexität der Fragestellung (↑Problemraum) begründet ist und der sich durch geschickten Einsatz von (etablierten) Strategien beheben lässt. Ein wesentlicher Baustein zur Verringerung dieser vermeidbaren Komplexität ist die Verwendung guter ↑Abstraktionen. Letztlich sind die ↑Programmierparadigmen genauso wie die in den letzten Jahrzehnten entwickelten Konzepte zur ↑Softwarearchitektur Strategien, diese Form der Komplexität weitestmöglich zu reduzieren. Eine grundlegende ↑Infrastruktur und sauberer Code (↑Clean Code) sind auf Ebene der Softwareentcklung weitere Strategien. Vgl. ↑unvermeidbare Komplexität.

Versionsverwaltung *version control system*, VCS; Software zur Verwaltung unterschiedlicher Versionen von Dateien und Programmen, die den Zugriff auf beliebige ältere als Versionen (↑Revision) gespeicherte Zustände ermöglicht. Gleichzeitig ein wichtiges Werkzeug für die Softwareentwicklung und wesentlicher Aspekt einer Projektinfrastruktur.

Literatur

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

[2] Frederick P. Brooks. *The Mythical Man Month*. Anniversary edition with four new chapters. Boston: Addison Wesley Longman, 1995.

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

[4] Robert C. Martin. *Clean Architecture. A Craftman's Guide to Software Structure and Design*. Boston: Prentice Hall, 2018.