Physikalische Chemie, Universität Rostock

Vorlesung: Wissenschaftliche Softwareentwicklung

Wintersemester 2024/25

Dr. habil. Till Biskup

— Glossar zu Lektion 31: „Finale furioso: Zusammenfassung und Ausblick“ —

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.

Aggregation *aggregation*, „klassische“ Form der ↑Zusammensetzung in der ↑objektorientierten Programmierung, definiert eine klare „*hat ein*“-Beziehung

Assoziation *association*, Interaktion von ↑Objekten/↑Klassen auf die Weise, dass ein Objekt/eine Klasse einen Service für ein anderes Objekt/eine andere Klasse bereitstellt.

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.

Busch, Wilhelm eigentlich Heinrich Christian Wilhelm Busch (1832–1908), einer der einflussreichsten humoristischen Dichter und Zeichner Deutschlands, bekannt für seine Bildergeschichten wie „Max und Moritz“, die ihn zu einem Pionier des Comics machten.

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

Datenformat digitales Speicherformat für Daten jeglicher Form. Grundsätzlich werden binäre und Textformate unterschieden. Während erstere meist mit deutlich geringerem Speicherbedarf auskommen, sind sie im Gegensatz zu letzteren nicht ohne Hilfsmittel lesbar. Textformate hingegen sind, ein beliebiger Texteditor vorausgesetzt, prinzipiell menschenlesbar.

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.

Infrastruktur personelle, sachliche und finanzielle Ausstattung, um ein angestrebtes Ziel zu er-

reichen. 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.

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, die den Menschen für gewöhnlich überfordern. ↑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.

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.

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.

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

Plattform Zusammenspiel aus Betriebssystem und Hardware-Architektur. Betriebssysteme können bis zu einem gewissen Grad unterschiedliche Hardware-Architekturen abstrahieren, so dass der gleiche Binärcode auf unterschiedlicher Hardware lauffähig ist, ohne neu kompiliert werden zu müssen.

plattformunabhängig Unabhängigkeit von einer spezifischen ↑Plattform. Neben Software sollten insbesondere ↑Datenformate plattformunabhängig sein, da nur so ein dauerhafter und unabhängiger Zugriff gewährleistet werden kann. Plattformunabhängigkeit hat sowohl eine Hardware- und Betriebssystemarchitekturals auch eine zeitliche Komponente. Alle drei großen Desktop-Betriebssysteme (Windows, Linux, macOS) sind in den Naturwissenschaften relativ weit verbreitet. Darüber hinaus findet die Entwicklung von Betriebssystemen auf der Zeitskala einer Archivierung schnell statt.

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

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.

Replizierbarkeit *replicability*, unabhängige Wiederholung der (Roh-)Datenerhebung, meist in Form von Experimenten und Beobachtungen, entsprechend nicht in jedem Fall durchführbar. Vgl. ↑Reproduzierbarkeit.

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.

Reproduzierbarkeit *reproducibility*, vollständige Wiederholbarkeit einer beschriebenen Datenverarbeitung und -Analyse. Ausgangspunkt sind existierende Daten, entsprechend sollte sie in jedem Fall möglich sein. Vgl. ↑Replizierbarkeit.

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*".

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.

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.

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.

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. ↑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. Vgl. ↑Zusammensetzung.

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.

Zusammensetzung *composition*, Wechselwirkung zwischen *unabhängigen* ↑Objekten, die ↑Kapselung bleibt im Gegensatz zur ↑Vererbung voll erhalten. Ein Objekt ist aus anderen Objekten zusammengesetzt. Die Objekte können anderweitig vollkommen unabhängig sein. Vgl. ↑Aggregation und ↑Assoziation.

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.