Physikalisch-Technische Bundesanstalt, Berlin (Adlershof)

Vorlesung: Wissenschaftliche Softwareentwicklung

2023/24

Dr. habil. Till Biskup

— Glossar zu Lektion 09: „Sauberer 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.

Broken-Window-Theorie in der Kriminologie und Soziologie diskutierte (und höchst umstrittene) Theorie: Eine kaputte Fensterscheibe an einem Haus, die zu lange nicht repariert wird, führt sehr schnell dazu, dass weitere Schäden hinzukommen, dass Leute beginnen, Müll an den falschen Stellen abzulagern, die Hauswand mit Graffitti zu beschmieren, etc. Anwendbar auf Qualität von Code: Offensichtliche Nachlässigkeit lädt nicht dazu ein, bei Änderungen selbst sauberer zu programmieren.

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

Code Review Gemeinsame Begutachtung eines Quellcode-Abschnitts eines Projektes durch mehrere Projektbeteiligte. Ein Code Review ist formaler als das ↑Pair Programming, außerdem wird beim Code Review in der Regel *nicht* programmiert, sondern nur über den Quellcode diskutiert. Ziele eines Code Reviews sind u.a.: Wissensvermittlung innerhalb des Teams, Sicherstellen eines gemeinsamen Kenntnisstands, Entwicklung von Ideen für die Weiterentwicklung. Eine Verbesserung des diskutierten Codes geht meist damit einher, allerdings sollte Kritik immer konstruktiv sein.

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.

DRY „*Don't repeat yourself*“, (nicht nur) in der angelsächsischen Programmierwelt verbreitetes Akronym und wichtige Regel für die Programmierung. Zentrales Programmierprinzip und Konzept für sauberen Code (↑Clean Code): Doppelungen im Code sind meist ein guter Hinweis darauf, dass eine ↑Abstraktion fehlt bzw. nicht erkannt wurde. Darüber hinaus sind Doppelungen im Code ein zentraler Grund für schlechte Qualität und Wartbarkeit.

Entwurfsmuster *design patterns*, erprobte und bewährte Lösungen für wiederkehrende Probleme in der Softwareentwicklung. Beschreibungen miteinander kommunizierender Objekte und Klassen, die maßgeschneidert sind, um ein generelles Entwurfsproblem in einem bestimmten Kontext zu lösen. Entwurfsmuster liefern eine (abstrakte) Beschreibung des dahinterstehenden Konzepts und haben meist einen etablierten Namen, der die Kommunikation erleichtert. Es gibt ganze Kataloge solcher Muster, und viele der ursprünglich be-

schriebenen Entwurfsmuster sind heute in vielen Programmiersprachen fest etabliert.

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 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).

intellektuelle Beherrschbarkeit *intellectual manageability*, nach Edsger Dijkstra das Hauptziel der Softwaretechnik – und letztlich des Projektmanagements. Unterschiedliche Lösungsansätze für ein Problem sind unterschiedlich gut intellektuell beherrschbar. Entsprechend ist die intellektuelle Beherrschbarkeit das zentrale Kriterium für die Entscheidung, welche Lösung für ein Problem bevorzugt wird.

monolithisch aus einem Stück bestehend; zusammenhängend und fugenlos

Pair Programming „Paarprogrammierung“, enge und aktive Zusammenarbeit zweier Programmierer bei der Programmierung von Software. Eine Person sitzt an der Tastatur und programmiert, während die andere aktiv über den entstehenden Code und das Problem nachdenkt und ggf. korrigierend eingreift. Auffallende Probleme werden sofort angesprochen und diskutiert. Die Rollen sollten häufig wechseln (mindestens innerhalb einer Stunde), idealerweise ebenso die Zusammensetzung der Paare.

Pfadfinderregel „Hinterlasse einen Ort immer in einem besseren Zustand als du ihn vorgefunden hast.“ Lässt sich auf die Softwareentwicklung anwenden und bezieht sich hier auf die kontinuierliche strukturelle Verbesserung von Code im Sinne seiner Lesbarkeit (↑Refactoring), um der ↑Software-Entropie vorzubeugen.

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.

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.

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. [1, S. 136]

Software-Entropie Die Qualität von Code nimmt über seine Lebenszeit hinweg ab. Grund sind unvermeidliche Anpassungen am Code und damit verbunden die Zunahme von Komplexität.

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.

Literatur

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