Lehr- und Übungsbuch Softwareentwicklung

Bearbeitet von
Peter Forbrig, Immo O. Kerner

1. Auflage 2004. Buch. 360 S. Hardcover
ISBN 978 3 446 22578 7
Format (B x L): 17,4 x 24,7 cm
Gewicht: 780 g

Zu Inhaltsverzeichnis

1 Einleitung

Juliane Benra

1.1 Einleitung

Software beeinflusst viele Bereiche unseres Lebens. Einige Anwendungen sind den meisten von uns im täglichen Leben schon sehr vertraut, wie etwa die Durchführung von Überweisungen von einem Bankterminal aus. Andere betreffen Anwendungen, mit denen nur wenige von uns zu tun haben, wie beispielsweise die Steuerung eines medizinischen Gerätes. Gemeinsam sind ihnen allgemeine Prinzipien, die die Entwicklung qualitätsgerechter Software unterstützen.

Gerade das „Weiche“ an Software macht sie so universell einsetzbar für die verschiedenen Gebiete. Dabei ist es ein langer Weg gewesen von den ersten Rechnern, die z. B. für wissenschaftliche Zwecke Differenzialgleichungen gelöst haben, bis hin zu modernen komplexen Softwaresystemen, wie sie uns heute im Arbeits- und Privatleben begegnen.

Dieses Kapitel soll einen Einblick in die Erstellung von professionellen Softwareprodukten geben. Das Ziel ist aber auch eine Sensibilisierung des Lesers für die speziellen Probleme, die diese Thematik mit sich bringt.

Zunächst erfolgt eine Einleitung mit der Darlegung wichtiger Begriffe, danach werden historische Zusammenhänge beschrieben. Anschließend wird die Disziplin Software Engineering kurz vorgestellt. Überlegungen zur Qualität sowie zur Wirtschaftlichkeit und Wiederverwendbarkeit folgen. Beendet wird das Kapitel mit einer Übersicht über weitere Faktoren, die die Softwareentwicklung beeinflussen.

1.1.1 Begriffsbestimmungen

Das Entwickeln von Software ist keineswegs eine einfache Tätigkeit, sondern birgt eine Vielzahl von Problemen. In diesem und den nächsten Kapiteln wird festgestellt werden, dass Kommunikation einen wesentlichen Bestandteil der Softwareentwicklung darstellt. Eine der ersten Schwierigkeiten besteht darin, Begriffe wie z. B. Software zu klären, so dass bei der Bewältigung der anstehenden Probleme in der Softwareentwicklung auf eine gemeinsame Begrifflichkeit zurückgegriffen werden kann.

Software ist mittlerweile fast ein „deutsches“ Wort. So findet man im Duden den Hinweis auf die elektronische Datenverarbeitung. Die weitere Erläuterung besagt, dass es sich um Programme und Anweisungen handelt. Software steht im Gegensatz zur so genannten Hardware, welche die fest installierte Technik bei Datenverarbeitungsanlagen darstellt. Auch im Brockhaus findet man eine Definition für Software: „Sammelbezeichnung für Programme, die für den Betrieb von Rechensystemen zur Verfügung stehen, einschließlich der zugehörigen Dokumentation.“

Innerhalb der Disziplin Informatik hat man sich schon vor langer Zeit Gedanken über eine Definition des zentralen Begriffes Software gemacht. So hat die Gesellschaft für

Informatik Arbeitsgruppen mit dem Auftrag gebildet, die Begriffsbildung zu vereinheitlichen. Dies ist bereits in den 80er-Jahren des letzten Jahrhunderts geschehen. Folgende Definition ist das Resultat dieser Überlegungen:

Software

Definition 1.1 Software

Software ist eine Menge von Programmen oder Daten zusammen mit begleitenden Dokumenten, die für ihre Anwendungen notwendig oder hilfreich sind /Hesse84, 204/.

Damit ist Software also keineswegs „nur“ ein Computerprogramm, sondern genauso die Dokumentation und alle zugehörigen Daten. Dazu gehört z. B. die Information, wie aus verschiedenen Programmkomponenten eine ablauffähige Software entsteht

Software-system

Definition 1.2 Softwaresystem

Mehrere Programme, Daten und Dokumentation, die zueinander in Beziehung stehen und ein gemeinsames Ganzes bilden, werden als Softwaresystem bezeichnet.

Als Spezialfall kann auch ein einzelnes Programm mit seinen Daten und der Dokumentation ein Softwaresystem darstellen.

Software-produkt

Definition 1.3 Softwareprodukt

Der Begriff Produkt steht für das in sich abgeschlossene Ergebnis eines Herstellungsprozesses, der im Allgemeinen durch einen Auftraggeber ausgelöst wurde. Ein Softwareprodukt ist ein Softwaresystem, das aus Auftraggebersicht betrachtet wird.

Software-entwicklung

Definition 1.4 Softwareentwicklung

Der Herstellungsprozess eines Softwareproduktes wird als **Softwareentwicklung** bezeichnet.

Daran beteiligt sein können Softwareentwickler, Auftraggeber, Nutzer, Betroffene, Investoren und andere. Im angloamerikanischen Sprachraum gibt es für Letztere den Sammelbegriff des *Stakeholders*.

Idealerweise werden die künftigen Benutzer in den Prozess mit einbezogen. Während die Softwareentwickler einen tiefen Einblick in die einzelnen technischen Komponenten des Softwaresystems haben, haben Auftraggeber und Benutzer normalerweise eine Sicht auf das Softwareprodukt, die eher an der Funktionalität und Ergonomie des Produktes orientiert ist.

Projekt

Definition 1.5 Projekt

Die Erstellung von Software erfolgt in Projekten. Der Begriff Projekt signalisiert, dass ein vorgegebenes Arbeitsziel in einer vorgegebenen Zeit unter Einsatz von vorhandenen, meist beschränkten Mitteln erarbeitet werden muss.

Die Notwendigkeit, ein solches Projekt wirtschaftlich erfolgreich durchzuführen, ist eines der Hauptprobleme bei Softwareprojekten.

1.1.2 Eigenschaften von Software

Softwareprodukte unterscheiden sich von anderen industriell gefertigten Produkten. Zunächst ist Software ein Produkt, das nicht unmittelbar wahrgenommen wird, sondern

nur durch den Ablauf auf einer konkreten Hardware und eventuell durch begleitende Texte (Programmtexte und Dokumentation) erschlossen werden kann.

Änderbarkeit

Software bedeutet „weiche“ Ware. Dadurch wird auf die (oft vermeintliche) schnelle **Änderbarkeit** von Software Bezug genommen. Sicher ist es schnell geschehen, ein paar Codezeilen zu verändern und den neuen Code auf einem Rechner ablaufen zu lassen. Das Problem bei vielen Systemen besteht darin, dass sie so programmiert und dokumentiert wurden, dass man vielfach nicht den Mut hat, an irgendeiner Stelle verändernd einzugreifen. Dies liegt daran, dass das System so komplex und unübersichtlich sein kann, dass man die Folgen einer Veränderung kaum noch einschätzen kann. Außerdem ist es mit einer reinen Veränderung vom Code nicht getan. Man muss das Ganze sinnvoll dokumentieren und die Qualität der erfolgten Veränderung sicherstellen, z. B. durch Durchführung von Tests (siehe Kapitel 5).

Wartung

Software unterliegt keinen Verschleißerscheinungen, wie es zum Beispiel die Hardware tut. Daher hat die **Wartung** von Software eine andere Aufgabe als bei vielen materiellen Produkten: Es geht bei der Softwarewartung um die Behebung von erkannten Fehlern und um notwendige Anpassungen an neue Rahmenbedingungen. Software überlebt die Hardware oft über mehrere Generationen. So ist heute noch Software im Einsatz, die in den 60er-Jahren des vergangen Jahrhunderts programmiert wurde.

1.1.3 Klassifizierung von Softwaresystemen

Es ist hilfreich, die verschiedenen Arten von Softwaresystemen zu klassifizieren. Dies kann dazu dienen, Gemeinsamkeiten von verschiedenen Softwarearten zu identifizieren. Dadurch können auch gemeinsame Schwierigkeiten bei der Entwicklung der Software einer Art herausgefunden werden und Problemlösungsstrategien für eine größere Anzahl von Softwaresystemen entwickelt werden. Die unten genannten Klassifizierungsmöglichkeiten sind keineswegs vollständig, sondern geben nur einen exemplarischen Eindruck von der Einteilung von Software in verschiedene Arten wieder.

Klassifizierung nach Größe

Verschiedene Maße können hierfür herangezogen werden. Beispiele sind etwa die Anzahl der Codezeilen oder die Anzahl der Monate, die Entwickler an einem Projekt gearbeitet haben. Letzteres wird in *Mitarbeitermonaten* ausgedrückt. Weitere Maße sind Kosten, Speicherplatzbedarf, Hardwareanforderungen und viele andere mehr.

Klassifizierung nach Anwendungsgebiet

Je nach Anwendungsgebiet ergeben sich unterschiedliche Schwerpunkte. Eine Anwendung für eine große Versicherung ist abhängig von der Verwaltung von Massendaten, wohingegen eine Anwendung in einer Automatisierungsstraße besondere Geschwindigkeitsaspekte erfüllen muss. Daher ist es sinnvoll, Softwaresysteme nach ihrem Anwendungsgebiet zu klassifizieren.

Klassifizierung nach Betriebssystemnähe

Anwendungssoftware

Systemsoftware

Software wird nicht nur für die klassischen Anwendungsfelder entwickelt (**Anwendungssoftware**), sondern es gibt auch Softwareentwicklung von **Systemsoftware**, die die Plattform für die Entwicklung der Anwendungssoftware zur Verfügung stellt. Klassisches Beispiel hierfür ist die Betriebssystementwicklung. Sie stellt andere Anforderungen an die Entwickler, z. B. hinsichtlich der Tiefe ihrer Kenntnisse über die verwendete Hardware.

Individual-software

Standard-software

Klassifizierung nach Spezialisierungsgrad

Software, die für einen speziellen Kunden genau zugeschnitten entwickelt wird (**Individualsoftware**), steht im Gegensatz zu **Standardsoftware**, die für eine größere Gruppe von Benutzern konzipiert ist und einen höheren Verbreitungsgrad hat. Der wesentliche Unterschied zwischen beiden Arten besteht darin, dass bei der Individualsoftware der Kunde während der Entwicklung mitreden kann (und soll), während bei der Entwicklung von Standardsoftware nur von allgemeinen Kundenwünschen und Bedürfnissen ausgegangen werden kann.

1.1.4 Kontrollfragen

1.1 Was ist der Unterschied zwischen Softwaresystem und Softwareprodukt?

1.2 Was verstehen Sie unter dem Begriff *Projekt*?

1.3 Was ist das Besondere bei der Wartung von Software?

1.4 Was ist der Grund dafür, dass man Software nach Anwendungsgebieten klassifiziert?

1.2 Softwareentwicklung früher und heute

In vielen Köpfen herrscht sie noch vor, die Vorstellung von einem einzelnen kontaktscheuen Entwickler, der in einer Garage geniale Hardware und zugehörige Programme entwickelt. Diese Vorstellung resultiert – neben der aktiven Unterstützung aus Hollywood und Co. – aus den frühen Jahren der Anwendung von Computern.

Zu Beginn des Computerzeitalters wurden Rechner wesentlich anders genutzt als heute. Der Entwickler von Software war bei ihrer Anwendung häufig anwesend.

Auch die Entwicklung von Software hatte einen anderen Stellenwert in einem EDV-Projekt (Hard- und Software). Heute ist die Software der wesentliche Zeit- und Kostenfaktor solcher Projekte. Sie muss erhöhten Qualitätskriterien genügen, weil ein Eingriff der Entwickler bei der Nutzung praktisch nicht mehr möglich ist. Dies bedeutet, dass die Entwicklung von Software heutzutage ingenieurmäßig organisiert werden muss, im Vergleich zu der eher „künstlerisch-kreativen“ Programmentwicklung der frühen Jahre.

1.2.1 Die frühen Jahre

Zu Beginn des Computereinsatzes waren Wissenschaft und Technik die wichtigsten Anwendungsgebiete. In der Regel wurden bereits bekannte mathematische Algorithmen in Rechnerprogramme übertragen.

Mit dem zunehmenden Einsatz von Rechnersystemen für nicht mehr rein numerisch-naturwissenschaftliche Probleme entstanden völlig neue Anforderungen an Softwareprodukte: Sie sollten leicht veränderbar sein, da sich die Randbedingungen für den Einsatz der Produkte häufig änderten, die Komplexität nahm zu, und es wurden nicht mehr „nur“ altbekannte Algorithmen umgesetzt.

„Als entscheidende Schwierigkeit erwies sich die Notwendigkeit, eine komplexe Aufgabenstellung in überschaubare Teilaufgaben zu zerlegen.“ /Suhr93, 31/.

Gleichzeitig änderte sich der Kreis der Personen, die an der Softwareentwicklung und -nutzung beteiligt waren. Waren in den frühen Jahren noch die späteren Benutzer der Software häufig auch die Programmierer (z. B. Wissenschaftler), vollzog sich später eine

Trennung beider Personengruppen. Dennoch sind auch heute die Softwareentwickler mitunter noch keine systematisch ausgebildeten Personen, was mit einer Vielzahl von Problemen einhergeht.

Softwarekrise

In den 60er-Jahren des 20. Jahrhunderts wuchsen sich diese Probleme langsam zu der so genannten **Softwarekrise** aus: Termine wurden nicht eingehalten, bzw. die ausgelieferten Produkte wiesen noch erhebliche Mängel auf. Dadurch entstanden große wirtschaftliche Schäden.

Bis heute existiert eine Vielzahl eigentlich veralteter Softwaresysteme, die nur schwierig zu ersetzen sind, da ihre Entwicklung noch nicht nach ingenieurmäßigem Vorgehen erfolgte.

1.2.2 Entstehung der Disziplin Software Engineering

Software Engineering

Ende der 60er-Jahre wird schließlich gefordert, dass ein ingenieurmäßiges Vorgehen für die Softwareentwicklung vonnöten sei. Dies war die Geburtsstunde der Disziplin **Software Engineering**.

Software bekommt eine zunehmende Bedeutung für alle Bereiche des täglichen Lebens. Dadurch haben sich Qualitätsanforderungen verändert, z. B. muss der Tatsache Rechnung getragen werden, dass die Benutzer von Software heute keineswegs Spezialisten im Umgang mit dem Computer sein müssen. Statt sich auf den korrekten Umgang mit dem Computer zu konzentrieren besteht die Anforderung, bei der durchzuführenden Tätigkeit durch den Computer unterstützt zu werden. Außerdem gibt es viele Anwendungen, die erhöhte Sicherheitsanforderungen stellen, da ein Nichtfunktionieren wirklich fatale Folgen haben könnte (z. B. Anwendungen in einem Atomkraftwerk oder Flugzeug).

Mittlerweile ist die Softwarekrise bereits eine geraume Zeit alt und es gibt viele, die behaupten, sie sei immer noch nicht überwunden. Manche Probleme erweisen sich gewissermaßen als inhärent bei der Bewältigung so komplexer Aufgabenstellungen, wie sie die Softwareentwicklung darstellt. Die Wünsche der Anwender richtig zu analysieren ist eines der größten Probleme. In Kapitel 3 wird dazu eine Hilfestellung gegeben. Aber auch der richtige Softwareentwurf basierend auf bereits bekannten Mustern ist keineswegs einfach. Dazu gibt Kapitel 4 eine Unterstützung.

Es ist aber offensichtlich, dass das Software Engineering eine lebendige Disziplin ist, die sich ständig weiterentwickelt und auch nach immer neuen, besseren Lösungen sucht.

1.2.3 Überlegungen zu den Kosten von Software

„Die Bedeutung von Korrektheit und Flexibilität, bzw. Anpassbarkeit von Software lässt sich am besten ermessen, wenn man die Entwicklungs- und Wartungskosten betrachtet.“ /Suhr93, 31/. Der Anteil der Kosten für Software an den Gesamtkosten von EDV-Projekten ist stetig ansteigend (nach EIA, /Suhr93, 31/). Daher ist es auch wirtschaftlich sinnvoll, das Augenmerk auf die Entwicklung von Software zu richten.

Der Zeitpunkt der Erkennung und Behebung von Fehlern innerhalb eines Entwicklungsprojektes ist dabei insofern von Interesse, als es gravierende Unterschiede hinsichtlich der entstehenden Behebungskosten gibt, je nachdem, wann der Fehler erkannt wird. Einfach formuliert: Umso später ein Irrtum/Fehler in einem Projekt entdeckt wird, desto eher entsteht ein Vielfaches der Kosten bei der Behebung (nach /Boehm76). Es ist also von hohem Interesse, Fehler und Irrtümer möglichst früh in einem Projekt aufzuklären.

1.2.4 Aktueller Stand

Im einundzwanzigsten Jahrhundert gibt es folgende besondere Herausforderungen für das Software Engineering (nach /Sommerville01, 13/:

Altlasten

- **Altlasten**

Viele große Softwaresysteme sind bereits vor vielen Jahren entwickelt worden. Mittlerweile ist eine Wartung dieser Software sehr schwierig, da sie zum Teil noch nicht ingenieurmäßig programmiert wurde und vielfach die ursprünglichen Entwickler nicht mehr in der Firma sind.

Heterogenität

- **Heterogenität**

Viele Systeme arbeiten mittlerweile als verteilte Systeme, die über ein Netzwerk verbunden sind. In diesen Netzwerken können unterschiedliche Computer mit unterschiedlicher Systemsoftware eingesetzt werden. Software muss in Zukunft sehr flexibel auf diese Anforderungen reagieren können.

Zeitdruck

- **Zeitdruck**

Viele Techniken des Software Engineering sind zeitaufwändig. Andererseits sind heute die Auslieferungszeiten für eine neue Software sehr kurz geworden. Da man aber solche komplexen Prozesse wie die der Softwareentwicklung nicht beliebig parallelisieren kann, wird der Druck auf Entwickler und Management größer oder geht zu Lasten der Qualität.

1.2.5 Kontrollfragen

1.5 Was ist das Ziel von Software Engineering?

1.6 Was verstehen Sie unter der „Softwarekrise“?

1.3 Software Engineering

Für die Entwicklung von Software ist auch heute noch ein gewisses Maß an Kreativität notwendig. Dennoch ist man sich mittlerweile einig, dass für ein gutes Softwareprodukt auch ingenieurmäßiges Arbeiten vonnöten ist. Aus dieser Erkenntnis ist die Disziplin Software Engineering entstanden.

Definition 1.6 Software Engineering (synonym **Softwaretechnik)**

Software Engineering ist eine Ingenieurdisziplin, die sich mit allen Aspekten der Softwareproduktion beschäftigt (nach /Sommerville01, 6f./).

Software Engineering Softwaretechnik

Ingenieurdisziplin bedeutet dabei, vor allen Dingen an einem gesicherten Weg zu einer konkreten Problemlösung interessiert zu sein. Am Ende des Entwicklungsprozesses muss eine gewisse Qualität des Softwareproduktes gesichert sein. Dafür werden unter Zuhilfenahme von angemessenen Werkzeugen bereits existierende Methoden genutzt. Außerdem sind Ingenieure sich der organisatorischen und finanziellen Grenzen ihres Tuns bewusst und kalkulieren diese Randbedingungen mit ein. Man beschränkt sich nicht nur auf den technischen Prozess, sondern betrachtet auch Dinge wie Projektmanagement, Entwicklung unterstützender Tools, Methoden etc. Es sind also alle Aspekte der Softwareproduktion betroffen.

Auf Grund einer solchen Modellvorstellung können schließlich Verfahren entwickelt werden, die bei der Projektabwicklung helfen. Es gibt eine ganze Reihe bekannter Vorgehensmodelle für die Entwicklung von Software, die im folgenden Abschnitt kurz beleuchtet werden. Etwas ausführlicher wird die Thematik dann in Kapitel 2 diskutiert.

1.3.1 Vorgehensmodelle

Lange vor der Schaffung von unterstützenden Werkzeugen für die Entwicklung von Software steht die Darstellung von **Vorgehensmodellen**.

Alle Vorgehensmodelle betrachten die Entwicklung von Software als einen Prozess, der die folgenden wesentlichen Aktivitäten enthalten muss (nach /Sommerville01, 8/):

- Spezifikation der Funktionalität der Software und aller Randbedingungen (Was soll die Software tun und was muss dabei berücksichtigt werden?)
- Entwicklung der Software gemäß der Spezifikation der Anforderungen
- Validierung der Software hinsichtlich der Kundenerwartungen
- Ermöglichen der Softwareevolution, falls sich Kundenanforderungen und/oder Randbedingungen ändern sollten.

Definition 1.7 Vorgehensmodell

Ein Vorgehensmodell stellt eine Beschreibung des Entwicklungsprozesses von Software dar.

Vorgehens-modell

Die ältesten, so genannten **Wasserfall-** oder **Phasenmodelle** stammen bereits aus der Zeit, in der die „**Softwarekrise**“ konstatiert wurde. Danach entstehen immer wieder neue Vorgehensmodelle, da Schwachpunkte bereits existierender Modelle entdeckt werden. Eine Vielzahl von Veränderungen an Vorgehensmodellen ist aus der Informatik selbst veranlasst; so hinterfragt man beispielsweise nach Einführung objektorientierter Analyse- und Designmethoden die bisherigen Vorgehensmodelle. Daraus folgen schließlich die **evolutionären Vorgehensmodelle**, nach denen eine Software in mehreren Evolutionsstufen entwickelt werden kann. Darin spiegelt sich die Idee, den Lernprozess aller Beteiligten während der Entwicklung von Software zur Entwicklung „besserer“ Software auszunutzen.

Wasserfall-, Phasenmodell

evolutionäre Vorgehens-modelle

Noch intensiver wird diese Idee beim **Extreme Programming** verfolgt, bei dem in sehr kurzen Iterationsschritten jeweils kleine Funktionalitätsveränderungen schnell durchgeführt werden.

Extreme Programming

Allen verschiedenen Vorgehensmodellen zu Grunde liegt die Vorstellung von einem **Softwarelebenszyklus**.

Definition 1.8 Softwarelebenszyklus (synonym **Software Life Cycle)**

Ein Softwarelebenszyklus basiert auf der Vorstellung, dass es bei der Softwareproduktion zyklisch wiederkehrende Aufgaben gibt. Ein Softwareprodukt wird geplant und entwickelt, geht in Betrieb und wird irgendwann nicht mehr alle Erfordernisse erfüllen. Danach muss erneut geplant und entwickelt werden, das heißt der Kreislauf beginnt von vorne.

Softwarele-benszyklus Software Life Cycle

1.3.1 Werkzeuge

Schon seit langem existiert die Wunschvorstellung, den Softwareentwicklungsprozess Werkzeug unterstützt zu betreiben.

Die Klasse von Werkzeugen (Tools), die hierbei eine Rolle spielen, sind die sogenannten CASE-Werkzeuge.

CASE-Werkzeug

Definition 1.9 CASE-Werkzeuge

Computer Aided Software Engineering (CASE) verwendet eine Klasse von Werkzeugen, die die ingenieurmäßige Erstellung von Software unterstützt.

Die Unterstützung des Entwicklungsprozesses durch diese Tools ist sehr unterschiedlich. Es gibt Werkzeuge, die lediglich die Möglichkeit bieten, Informationen grafisch geschickt aufzubereiten, so dass sie der Notation eines bestimmten Vorgehensverfahrens entsprechen. Andere Werkzeuge unterstützen nur einen bestimmten Teil bei der Softwareentwicklung: die Programmierung, den Entwurf etc. Idealerweise werden durch ein CASE-Werkzeug die gesamten Arbeitsschritte der Softwareentwicklung unterstützt und insbesondere Konsistenzprüfungen zwischen einzelnen Projektdokumenten vorgenommen. Dies bedeutet, dass die einzelnen Entwicklungsdokumente in sich widerspruchsfrei sein müssen. Einen Überblick über eine Reihe bestehender Werkzeuge gibt das Kapitel 10.

Soweit die Entwicklung im CASE-Bereich bereits gediehen ist, die Softwareentwicklung „auf Knopfdruck“ bleibt in den meisten Fällen noch der Wunschtraum.

Eine große Rolle spielen mittlerweile auch Werkzeuge, die es ermöglichen, bereits geschriebene und getestete Software zu finden und in neue Zusammenhänge einzubinden. Hier geht es um das weite Feld der Wiederverwendbarkeit und des Wissensmanagements, die beide ohne Werkzeugunterstützung nicht denkbar sind.

1.3.2 Kontrollfragen

1.7 Was leistet ein CASE-Werkzeug?
1.8 Wozu dient ein Vorgehensmodell?
1.9 Was gehört zum Lebenszyklus von Software?

1.4 Qualität

Heutzutage erwartet man von einem Softwareprodukt nicht mehr nur die reine Funktionalität (siehe Kapitel 5). Es werden auch Anforderungen hinsichtlich Ergonomie (siehe dazu Kapitel 11), Effizienz und anderer Aspekte gestellt.

Dabei können schon vermeintlich geringe Defekte an einem Softwareprodukt den Anwender in die Irre führen.

Beispiel 1.1 (nach /Balzert00, 30/)

In den Jahren zwischen 1977 und 1994 ist die Defektrate auf den vermeintlich sehr guten Wert von 0,2 … 0,05 Defekten pro 1000 Zeilen Quellcode gesenkt worden (um ungefähr das 100fache, verglichen mit den Vorjahren). Eine Defektrate von beispielsweise 0,1% bedeutet jedoch:

Pro Jahr ca. 300 versagende Herzschrittmacher

Pro Woche ca. 500 Fehler in einer medizinischen Operation
Pro Tag 16000 verlorene Briefe
Pro Stunde 22000 Schecks, die falsch verbucht werden.

Es ist daher leicht einzusehen, dass die Qualitätssicherung einer Software ein wichtiger Aspekt bei ihrer Entwicklung ist.

Definition 1.10 Qualität nach DIN55359 Teil 11

Qualität ist die Gesamtheit von Eigenschaften und Merkmalen eines Produktes oder einer Tätigkeit, die sich auf deren Eignung zu Erfüllung gegebener Erfordernisse bezieht.

Qualität

1.4.1 Qualitätsmerkmale für Software

Jeder kennt die Stiftung Warentest, die Qualitätsurteile für unterschiedliche Produkte fällt. Solchermaßen durchgeführte Tests sind nur dann seriös, wenn – wie im Falle von Stiftung Warentest – genau aufgeschlüsselt wird, welche Kriterien in welchem Maße in die Beurteilung eingeflossen sind.

Auch bei der Beurteilung von Softwarequalität bricht man den Qualitätsbegriff auf bestimmte Merkmale herunter. Dabei ist es erstrebenswert, die Qualitätsmerkmale eindeutig messen zu können.

Wesentliche Qualitätsmerkmale sind zum Beispiel (nach /Pomberger93, 10f./):

- **Korrektheit**, die Eigenschaft, dass die Spezifikationsvorgabe erfüllt wird
- **Zuverlässigkeit**, die Eigenschaft, dass ein Programm nicht nur korrekt arbeitet, sondern auch verfügbar ist (Vermeidung von Ausfallzeiten)
- **Benutzerfreundlichkeit**, das Programm soll zum Einsatzzweck adäquat sein, seine Bedienung soll gut erlernbar sein und sich robust verhalten (z. B. gegen Fehlbedienung)
- **Wartungsfreundlichkeit**, Eignung für die Lokalisierung von Fehlern und deren Korrektur bzw. zur Veränderung des Programms
- **Effizienz**, die bestmögliche Ausnutzung von Ressourcen, wie z. B. Prozessorzeit oder Speicherbedarf
- **Portabilität**, die Eignung, es leicht auf andere Rechner zu übertragen.

Korrektheit

Zuverlässigkeit

Benutzerfreundlichkeit

Wartungsfreundlichkeit

Effizienz

Portabilität

Qualitätsmerkmale sind durchaus nicht unabhängig voneinander. Es ist möglich, dass die Verbesserung hinsichtlich eines Merkmales eine Verschlechterung hinsichtlich eines anderen bewirkt. Beispielsweise kann eine Erhöhung von Benutzerfreundlichkeit zu langsamerer und damit ineffizienterer Software führen. Darüber hinaus hat die Qualitätssicherung Auswirkungen auf die Kosten- und Zeitplanung eines Projektes.

Um Qualität zu sichern, greift man zum einen auf konstruktive, zum anderen auf analytische Maßnahmen zurück (nach /Knöll96, 16/). **Konstruktive Qualitätssicherung** wird etwa betrieben, wenn Verfahren für ein Projekt vorgeschrieben sind und die Einhaltung dieser Verfahren beachtet wird. **Analytische Qualitätssicherung** erfolgt beispielsweise durch die Durchführung von Tests.

konstruktive Qualitätssicherung

analytische Qualitätssicherung

1.4.1 ISO 9000 und ähnliche Konzepte

Qualitäts-management

ISO 9001…9004

Total Quality Management (TQM)

Zu den bekanntesten Bestrebungen **Qualitätsmanagement** zu betreiben gehören die **ISO 9001…9004** und das **Total Quality Management** (**TQM**). Beiden Bestrebungen ist es gemein, dass deren Zielsetzung über die Bewältigung einzelner Projekte hinausgeht. In der ISO 9001 orientiert man sich unternehmensweit an den Qualitätsanforderungen des Entwicklungsunternehmens und dessen Unternehmenszielen. Im TQM wird zusätzlich auf die Befriedigung der Kundenbedürfnisse besonderer Wert gelegt. Darüber hinaus wird im TQM eine kontinuierliche Verbesserung der Unternehmensleistung angestrebt, indem das Qualitätsmanagement ständig optimiert wird.

1.4.2 Kontrollfragen

1.10 Was ist Effizienz?

1.11 Nennen Sie eine Maßnahme zur analytischen Qualitätssicherung.

1.5 Wirtschaftlichkeit

Damit die angestrebten Ziele eines Projektes eingehalten werden können, bedarf es eines Projektmanagements.

1.5.1 Projektmanagement

Mittlerweile gibt es eine Vielzahl von möglichen Strategien ein Projekt zu managen. Gerade dies ist für viele Industriepartner eine Schwierigkeit in sich.

„Viele Industriepartner wünschen sich (...) eine methodische Unterstützung für den Vergleich verschiedener Ansätze zur Softwareentwicklung. Nur so kann für ein bestimmtes Projekt schnell und kostengünstig die passende Entwicklungsstrategie gefunden werden.“ (nach /Deifel99, 34/).

Projekt-management

Auf jeden Fall muss das **Projektmanagement** wesentliche Sünden verhindern, die zu einem Misserfolg führen (nach /McConnell02/ und /Pomberger93, 297/):

- Keine Planung durchzuführen
- Keine Projektorganisation zu bestimmen, wie z. B. Projektstandards
- Nicht alle Projektaktivitäten mit einzurechnen, z. B. Meetings vergessen
- Keine Risiken mitzukalkulieren, z. B. keine Personenausfälle einzurechnen
- Immer gleich vorzugehen, ohne besondere Randbedingungen zu berücksichtigen
- Verfahren unkritisch zu übernehmen, ohne deren Übertragbarkeit auf die eigenen Gegebenheiten zu prüfen
- Zuzulassen, dass sich Planung und Wirklichkeit auseinander entwickeln, z. B. Diskrepanz zwischen Dokumentation und Wirklichkeit
- Nicht zu merken, dass Planung und Wirklichkeit auseinander klaffen, z. B. hinsichtlich der Kostenentwicklung, der Qualität oder des Fortschrittes
- Zu viele Details zu früh zu planen, sich dadurch zu verzetteln und viele Arbeitsergebnisse mehrfach zu erarbeiten
- Vorzusehen, eventuelle Rückstände später aufzuholen
- Nicht aus alten Planungsfehlern zu lernen.