Einführung

1

ÜBERBLICK

Einführung

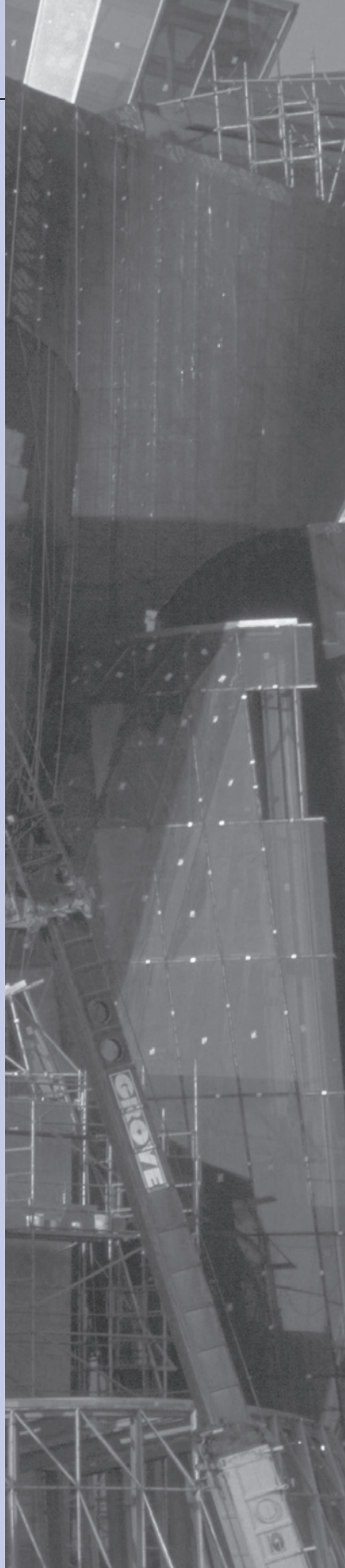
Die Lernziele dieses Kapitels sind, Sie mit dem Software-Engineering vertraut zu machen und Ihnen ein Gerüst für das Verständnis des restlichen Buches zu geben. Wenn Sie dieses Kapitel gelesen haben, werden Sie

- verstehen, was Software-Engineering ist und warum es wichtig ist;
- verstehen, dass die Entwicklung von unterschiedlichen Typen von Softwaresystemen auch unterschiedliche Software-Engineering-Techniken erforderlich machen kann;
- die ethischen und beruflichen Aspekte verstehen, die für Softwareentwickler wichtig sind;
- in drei Systeme verschiedener Typen eingeführt, die als Beispiele im gesamten Buch benutzt werden.

Wir können die moderne Welt ohne Software nicht aufrechterhalten. Staatliche Infrastrukturen und Versorgungseinrichtungen werden von computerbasierten Systemen gesteuert und die meisten elektrischen Produkte enthalten einen Computer und Steuerungssoftware. Industrielle Fertigungs- und Verteilungsprozesse sind – wie auch das Finanzsystem – vollständig computergestützt. Unterhaltung, einschließlich der Musikindustrie, Computerspiele, Film und Fernsehen, ist softwareintensiv. Daher ist Software-Engineering für das Funktionieren nationaler und internationaler Gesellschaften von entscheidender Bedeutung.

Softwaresysteme sind abstrakt und nicht greifbar. Sie werden weder durch Materialeigenschaften noch durch physikalische Gesetze oder durch Herstellungsprozesse beschränkt. Dies vereinfacht das Software-Engineering, da es keine natürlichen Grenzen des Potenzials von Software gibt. Jedoch können Softwaresysteme wegen der fehlenden physischen Einschränkungen schnell äußerst komplex, schwer durchschaubar und teuer bei Änderungen werden.

Es gibt viele unterschiedliche Arten von Softwaresystemen, von einfachen **eingebetteten Systemen** bis hin zu komplexen, weltweiten Informationssystemen. Es ist sinnlos nach Notationen, Methoden oder Techniken des Software-Engineerings zu suchen, weil unterschiedliche Softwaretypen unterschiedliche Ansätze erforderlich machen. Die Entwicklung eines Informationssystems für ein Unternehmen unterscheidet sich vollständig von der Entwicklung eines Steuergeräts für wissenschaftliche Instrumente. Und keines dieser Systeme hat viel gemeinsam mit einem grafikintensiven Computerspiel. Alle diese Anwendungen benötigen Software-Engineering, doch sie brauchen nicht alle die gleichen Software-Engineering-Techniken.



Es gibt immer noch viele Berichte von misslungenen Softwareprojekten und „Softwarefehlern“. Software-Engineering wird als unangemessen für moderne Softwareentwicklung kritisiert. Meiner Meinung nach resultieren jedoch viele dieser sogenannten Softwarefehler aus zwei Faktoren:

1. *Steigende Anforderungen:* In dem Maße, in dem neue Techniken des Software-Engineerings uns helfen größere, komplexere Systeme zu bauen, ändern sich auch die Anforderungen. Die Systeme müssen schneller gebaut und ausgeliefert werden; größere, noch komplexere Systeme werden benötigt; Systeme müssen ein Leistungsvermögen aufweisen, das zuvor für unmöglich gehalten wurde. Mit den vorhandenen Methoden des Software-Engineerings kann dies nicht bewältigt werden, sodass neue Software-Engineering-Techniken erarbeitet werden müssen, um diese neuen Anforderungen zu erfüllen.

2. *Geringe Erwartungen:* Es ist relativ einfach, Computerprogramme ohne die Anwendung von Software-Engineering-Methoden und -Techniken zu schreiben. Viele Unternehmen wurden im Zuge der Weiterentwicklung ihrer Produkte und Dienstleistungen in die Softwareentwicklung hineingetrieben. Sie benutzen die Methoden des Software-Engineerings nicht in ihrem Arbeitsalltag. Infolgedessen ist ihre Software häufig teurer und weniger zuverlässig als sie sein sollte. Das Fachgebiet Software-Engineering bedarf einer besseren Ausbildung und Schulungen, um diesem Problem zu begegnen.

Softwareentwickler können mit Recht stolz darauf sein, was sie erreicht haben. Natürlich haben wir noch Probleme bei der Entwicklung von komplexer Software, doch ohne Software-Engineering hätten wir das Weltall nicht erforscht, müssten ohne das Internet und ohne moderne Telekommunikation auskommen. Jede Art von Fortbewegung wäre gefährlicher und teurer. Software-Engineering hat schon viel bewegt und ich bin sicher, dass dieser Beitrag im 21. Jahrhundert sogar noch größer sein wird.

Geschichte des Software-Engineerings

Der Begriff „Software-Engineering“ wurde das erste Mal 1968 auf einer Konferenz vorgeschlagen, die sich mit der „Softwarekrise“ beschäftigte, wie diese seinerzeit bezeichnet wurde (Naur und Randell, 1969). Es wurde damals offensichtlich, dass individuelle Ansätze zur Programmentwicklung sich nicht auf große und komplexe Softwaresysteme übertragen ließen. Diese waren unzuverlässig, kosteten mehr als erwartet und wurden verspätet ausgeliefert.

Während der 1970er und 1980er Jahre wurde eine Vielzahl neuer Software-Engineering-Techniken und -methoden entwickelt, wie beispielsweise strukturierte Programmierung, **Datenkapselung** (*Information Hiding*) und **objektorientierte Entwicklung**. Werkzeuge und Standardnotationen wurden erstellt und werden nun ausgiebig genutzt.

http://www.SoftwareEngineering-9.com/Web/History

Link

1.1 Professionelle Softwareentwicklung

Viele Menschen schreiben Programme. Geschäftsleute schreiben Tabellenkalkulationsprogramme, um sich die Arbeit zu erleichtern, Wissenschaftler und Ingenieure schreiben Programme, um ihre Versuchsdaten zu verarbeiten, und Privatleute schreiben Programme für ihre eigenen Interessen und zum Vergnügen. Überwiegend ist die Softwareentwicklung jedoch eine professionelle Tätigkeit, bei der Software für bestimmte Geschäftszwecke erstellt wird, zur Einbindung in andere Geräte oder als Softwareprodukte wie Informationssysteme, CAD-Systeme usw. Professionelle Software, die außer vom Programmierer auch von anderen Menschen verwendet werden soll, wird normalerweise eher von Teams als von einzelnen Personen entwickelt. Diese Software wird während ihrer gesamten Lebensdauer gewartet und angepasst.
Software-Engineering soll eher die professionelle Softwareentwicklung als das individuelle Programmieren unterstützen. Es beinhaltet Techniken zur Programmspezifikation, für den Entwurf und zur Weiterentwicklung – nichts, was normalerweise für die private Softwareentwicklung von Belang ist. Um Ihnen zu helfen, einen umfassenden Blick dafür zu gewinnen, worum es beim Software-Engineering geht, habe ich eine Liste von häufigen Fragen und Antworten in ▶ Abbildung 1.1 zusammengestellt.

Frage	Antwort
Was ist Software?	Computerprogramme und zugehörige Dokumentation. Softwareprodukte können für einen bestimmten Kunden oder für den freien Markt entwickelt werden.
Was sind die Eigenschaften von guter Software?	Gute Software sollte dem Nutzer die geforderte Funktionalität und Performanz liefern und sollte wartbar, verlässlich und nützlich sein.
Was versteht man unter Software-Engineering?	Software-Engineering ist eine technische Disziplin, die sich mit allen Aspekten der Softwareherstellung beschäftigt.
Welches sind die grundlegenden Aktivitäten des Software-Engineerings?	Softwarespezifikation, Softwareentwicklung, Softwarevalidierung und Softwareweiterentwicklung.
Worin liegt der Unterschied zwischen Software-Engineering und Informatik?	Informatik konzentriert sich auf Theorie und Grundlagen; beim Software-Engineering geht es um die Entwicklung in der Praxis und die Herstellung nützlicher Software.
Worin liegt der Unterschied zwischen Software-Engineering und Systems-Engineering?	Systems-Engineering beschäftigt sich mit allen Aspekten computerbasierter Systementwicklung, darunter Hardware-, Software- und Verfahrensentwicklung. Software-Engineering ist ein Teil dieses Prozesses.
Worin liegen die größten Herausforderungen für das Software-Engineering?	Der erfolgreiche Umgang mit zunehmender Vielfalt, mit Forderungen nach verkürzten Lieferzeiten und mit der Entwicklung vertrauenswürdiger Software.
Was kostet das Software-Engineering?	Etwa 60 % der Kosten sind Entwicklungskosten, 40 % sind Kosten für Tests. Bei Auftragssoftware übersteigen die Kosten für die Weiterentwicklung oft die für die eigentliche Entwicklung.

Abbildung 1.1: Häufig gestellte Fragen zum Thema Software.

Frage	Antwort
Welches sind die besten Techniken und Methoden des Software-Engineerings?	Da alle Softwareprojekte professionell verwaltet und entwickelt werden müssen, sind unterschiedliche Techniken für verschiedene Systemarten angemessen. Zum Beispiel sollten Spiele immer mithilfe einer Folge von Prototypen entwickelt werden, wohingegen sicherheitskritische Steuerungssysteme eine vollständige und analysierbare Spezifikation zur Entwicklung erfordern. Man kann also nicht sagen, dass eine Methode besser als eine andere ist.
Wie hat das Web das Software-Engineering verändert?	Das Web hat zur Verfügbarkeit von Softwareleistungen geführt und der Möglichkeit, hochgradig verteilte servicebasierte System zu entwickeln. Die Entwicklung von webbasierten Systemen hat zu wichtigen Fortschritten bei Programmiersprachen und der Wiederverwendung von Software geführt.

Abbildung 1.1: Häufig gestellte Fragen zum Thema Software. *(Forts.)*

Viele Menschen denken, dass „Software“ nur ein anderes Wort für „Computerprogramm“ ist. Wenn wir jedoch über Software im Zusammenhang mit Software-Engineering sprechen, dann sind damit nicht nur die Programme selbst gemeint, sondern auch die dazugehörige Dokumentation und die Konfigurationsdaten, die erforderlich sind, damit diese Programme richtig funktionieren. Ein professionell entwickeltes Softwaresystem ist oft mehr als ein einzelnes Programm. Das System besteht normalerweise aus mehreren separaten Programmen und Konfigurationsdateien, die zur Einrichtung dieser Programme verwendet werden. Es kann die Systemdokumentation enthalten, die die Systemstruktur beschreibt, die Benutzerdokumentation, die die Anwendung des Programms erklärt, und Websites, von denen der Benutzer die neuesten Produktinformationen herunterladen kann.

Dies ist einer der wichtigsten Unterschiede zwischen professioneller und amateurhafter Softwareentwicklung. Wenn man ein Programm für sich selbst schreibt, dann wird es niemand anderes nutzen und man muss sich nicht die Mühe machen, Programmanleitungen zu schreiben, den Programmentwurf zu dokumentieren usw. Schreibt man dagegen Software, die andere Menschen benutzen und andere Programmierer ändern werden, dann muss man gewöhnlich zusätzliche Information sowie den Programmcode zur Verfügung stellen.

Softwareentwickler beschäftigen sich mit der Entwicklung von Softwareprodukten (d. h. mit Software, die an einen Kunden verkauft werden kann). Es gibt zwei grundlegende Arten von Softwareprodukten:

1. *Generische Produkte* sind eigenständige Systeme, die von einem Softwareentwicklungsunternehmen hergestellt wurden und auf dem freien Markt an jeden Kunden verkauft werden, der es sich leisten kann. Beispiele für diese Art von Produkten ist Software für PCs, wie Datenbanken, Textverarbeitungsprogramme, Grafikpakete oder Projektverwaltungswerkzeuge. Dazu gehören auch sogenannte vertikale Anwendungen, die für spezielle Zwecke entworfen wurden, wie Bibliotheksinformationssysteme, Abrechnungssysteme oder Systeme zur Verwaltung von zahnärztlichen Aufzeichnungen.

2 *Angepasste (oder bestellte) Produkte* sind Systeme, die im Auftrag eines bestimmten Kunden hergestellt werden. Ein Softwareanbieter entwickelt die Software speziell für diesen Kunden. Beispiele für solche Auftragssoftware sind Steuerungssysteme für elektronische Geräte, Systeme zur Unterstützung eines bestimmten Geschäftsprozesses und Flugsicherungssysteme.

Ein wichtiger Unterschied zwischen diesen Softwaretypen ist, dass bei generischen Produkten das Unternehmen, das die Software entwickelt hat, auch die Spezifikation der Software selbst bestimmt. Bei Auftragsprodukten wird die Spezifikation normalerweise von dem Unternehmen entwickelt und kontrolliert, das die Software kauft. Die Softwareentwickler müssen sich an diese Spezifikation halten.

Die Unterscheidung zwischen diesen Systemproduktttypen verwischt jedoch zunehmend. Es werden heutzutage immer mehr Systeme mit einem generischen Produkt als Grundlage gebaut, das dann an die Kundenbedürfnisse angepasst wird. **ERP-Systeme** (*Enterprise Resource Planning System*) wie das SAP-System sind beste Beispiele für diesen Ansatz. Hier wird ein großes und komplexes System für ein Unternehmen angepasst, indem Informationen über Geschäftsregeln und -prozesse, erforderliche Berichte usw. integriert werden.

Wenn wir über die Qualität von professioneller Software sprechen, dann müssen wir berücksichtigen, dass die Software von anderen Menschen als ihren Entwicklern genutzt und verändert wird. Qualität betrifft deshalb nicht nur das, was die Software macht. Vielmehr muss sie das Verhalten der Software, während diese ausgeführt wird, sowie die Struktur und Organisation der Systemprogramme und der zugehörigen Dokumentationen beinhalten. Dies schlägt sich in sogenannten Qualitäts- oder nichtfunktionalen Softwaremerkmalen nieder. Beispiele dieser Merkmale sind die Antwortzeit der Software auf die Anfrage eines Benutzers und die Verständlichkeit des Programmcodes.

Die konkrete Menge dieser Attribute, die man von einem Softwaresystem erwarten könnte, hängt offensichtlich von seiner Anwendung ab. Ein Banksystem muss sicher sein, ein interaktives Spiel muss reaktionsfähig sein, ein Fernsprechvermittlungssystem muss zuverlässig sein und so weiter. Daraus kann man eine Reihe allgemeiner Merkmale herleiten, die in ▶ Abbildung 1.2 aufgelistet sind und von denen ich glaube, dass sie die wesentlichen Charakteristika eines professionellen Softwaresystems darstellen.

Produkteigenschaft	Beschreibung
Wartbarkeit (*maintainability*)	Software sollte so geschrieben werden, dass sie weiterentwickelt werden kann, um sich verändernden Kundenbedürfnissen Rechnung zu tragen. Das ist ein entscheidendes Merkmal, weil Softwareanpassungen eine unvermeidliche Anforderung einer sich verändernden Geschäftsumgebung sind.
Verlässlichkeit (*dependability*) und Informationssicherheit (*security*)	Die Softwareverlässlichkeit umfasst eine ganze Reihe von Merkmalen, darunter **Zuverlässigkeit** (*reliability*), **Informationssicherheit** (*security*) und **Betriebssicherheit** (*safety*). Verlässliche Software sollte keinen körperlichen oder wirtschaftlichen Schaden verursachen, falls das System ausfällt. Böswillige Benutzer sollten nicht in der Lage sein, auf das System zuzugreifen oder es zu beschädigen.
Effizienz (*efficiency*)	Software sollte nicht verschwenderisch mit Systemressourcen wie Speicher und Prozessorkapazität umgehen. Effizienz umfasst somit Reaktionszeit, Verarbeitungszeit, Speichernutzung usw.
Akzeptanz (*acceptability*)	Software muss von den Benutzern akzeptiert werden, für die sie entwickelt wurde. Das bedeutet, dass sie verständlich, nützlich und kompatibel mit anderen Systemen sein müssen, die diese Benutzer verwenden.

Abbildung 1.2: Wesentliche Merkmale guter Software.

1.1.1 Software-Engineering

Software-Engineering ist eine technische Disziplin, die sich mit allen Aspekten der Softwareherstellung beschäftigt, von den frühen Phasen der Systemspezifikation bis hin zur Wartung des Systems, nachdem der Betrieb aufgenommen wurde. In dieser Definition gibt es zwei Schlüsselbegriffe:

1. *Technische Disziplin:* Techniker bringen Dinge zum Laufen. Sie benutzen wenn möglich Theorien, Methoden und Werkzeuge. Sie wählen sie jedoch bewusst aus und versuchen immer, Lösungen für Probleme zu finden, auch wenn es für diese Probleme keine anwendbaren Theorien oder Methoden gibt. Techniker erkennen auch an, dass sie mit organisatorischen und finanziellen Beschränkungen arbeiten müssen. Daher suchen sie ihre Lösungen innerhalb dieser Beschränkungen.

2. *Alle Aspekte der Softwareherstellung:* Software-Engineering beschäftigt sich nicht nur mit den technischen Prozessen der Softwareentwicklung, sondern umfasst auch Aktivitäten wie die Softwareprojektverwaltung und die Entwicklung von Werkzeugen, Methoden und Theorien, die die Softwareherstellung unterstützen.

Im Ingenieurwesen geht es darum, Ergebnisse in der geforderten Qualität innerhalb des Zeitplans und des Budgets zu erhalten. Dazu müssen häufig Kompromisse eingegangen werden – Ingenieure dürfen dann keine Perfektionisten sein. Jemand, der Programme zur Eigenanwendung schreibt, kann sich hingegen so viel Zeit für die Programmentwicklung nehmen, wie er will.
Im Allgemeinen arbeiten Softwareentwickler vorwiegend systematisch und organisiert, weil dies oft die effektivste Art ist, qualitativ hochwertige Software herzustellen. Im

Software-Engineering geht es jedoch hauptsächlich darum, die am besten geeignete Methode für die jeweiligen Umstände zu wählen. Ein kreativeres, weniger formelles Verfahren kann sich unter bestimmten Umständen als effektiv herausstellen. Die weniger formelle Entwicklung ist besonders für die Entwicklung von webbasierten Systemen angemessen, weil diese eine Mischung aus Software und Grafikdesign verlangt.
Software-Engineering ist aus zwei Gründen wichtig:

1. Individuen und die Gesellschaft sind immer mehr auf fortschrittliche Softwaresysteme angewiesen. Wir müssen zuverlässige und vertrauenswürdige Systeme wirtschaftlich und schnell herstellen.

2. Auf lange Sicht ist es gewöhnlich billiger, Methoden und Techniken des Software-Engineerings für Softwaresysteme zu verwenden, anstatt das Programm so zu schreiben, als sei es ein privates Programmierprojekt. Für die meisten Systemtypen bilden die Kosten für Änderungen, wenn die Software bereits im Einsatz ist, den größten Kostenblock.

Der systematische Ansatz, der beim Software-Engineering eingesetzt wird, wird manchmal **Softwareprozess** genannt. Ein Softwareprozess ist eine Folge von Aktivitäten, die zur Herstellung eines Softwareprodukts führen. Es gibt vier grundlegende Aktivitäten, die alle Softwareprozesse gemeinsam haben. Diese Aktivitäten sind:

1. Softwarespezifikation: Kunden und Entwickler definieren die zu produzierende Software und die Rahmenbedingungen für ihren Einsatz.

2. Softwareentwicklung: Die Software wird entworfen und programmiert.

3. Softwarevalidierung: Die Software wird überprüft um sicherzustellen, dass sie den Kundenanforderungen entspricht.

4. Softwareevolution: Die Software wird weiterentwickelt, damit sie die sich ändernden Bedürfnisse der Kunden und des Marktes widerspiegelt.

Unterschiedliche Systemtypen erfordern unterschiedliche Entwicklungsprozesse. So muss z. B. in einem Flugzeug die Echtzeitsoftware vollständig spezifiziert werden, bevor die Entwicklung beginnt. In E-Commerce-Systemen werden die Spezifikation und das Programm in der Regel gemeinsam entwickelt. Folglich können diese allgemeinen Aktivitäten für unterschiedliche Softwaretypen auf unterschiedliche Arten organisiert und auf verschiedenen Detailebenen beschrieben werden – abhängig vom Softwaretyp, der entwickelt wird. Ich beschreibe Softwareprozesse detaillierter in *Kapitel 2*.
Software-Engineering ist sowohl mit der Informatik als auch mit dem **Systems-Engineering** verbunden:

1. In der Informatik geht es um die Theorien und Methoden, die Computern und Softwaresystemen zugrunde liegen, während sich Software-Engineering mit den praktischen Problemen der Softwareherstellung beschäftigt. Einige Kenntnisse auf dem Gebiet der Informatik sind für Softwareentwickler auf dieselbe Weise unverzichtbar, wie Grundlagenkenntnisse der Physik für Elektrotechniker unerlässlich sind. Die Theorie der Informatik ist jedenfalls insbesondere auf relativ kleine Programme anwendbar. Elegante Theorien der Informatik können oft nicht auf große, komplexe Probleme angewendet werden, die nach einer Softwarelösung verlangen.

2 Systems-Engineering beschäftigt sich mit allen Aspekten der Entwicklung und Evolution von komplexen Systemen, in denen Software eine wesentliche Rolle spielt. In der Systementwicklung geht es somit um Hardwareentwicklung, Vorgehens- und Verfahrensentwurf, Verteilung des Systems im Einsatz sowie Software-Engineering. Systementwickler beteiligen sich an der Spezifikation des Systems, der Definition der Gesamtarchitektur und an der Integration der verschiedenen Einzelteile, um das fertige System zu schaffen. Ihr Interesse gilt weniger der Entwicklung der Systemkomponenten (Hardware, Software usw.).

Im nächsten Abschnitt stelle ich eine Vielzahl von Softwaretypen vor. Es gibt keine universelle Methode oder Technik des Software-Engineerings, die für all diese Arten anwendbar ist. Doch drei allgemeine Probleme betreffen viele unterschiedliche Softwaretypen:

1 *Heterogenität:* Systeme müssen verstärkt als **verteilte Systeme** in Netzwerken arbeiten, die aus verschiedenen Computertypen und mobilen Geräten bestehen. Die Software muss nicht nur auf Vielzweckcomputern, sondern möglicherweise auch auf Mobiltelefonen ausgeführt werden. Man muss häufig neue Software in ältere Systeme integrieren, die in verschiedenen Programmiersprachen geschrieben sind. Die Herausforderung hier besteht also darin, Techniken zu entwickeln, mit denen verlässliche Software hergestellt werden kann, die flexibel genug ist, um mit dieser Heterogenität umzugehen.

2 *Geschäftliche und soziale Veränderungen:* Geschäftsleben und Gesellschaft ändern sich unglaublich schnell, ebenso wie sich neu entstehende Wirtschaftszweige entwickeln und neue Technologien verfügbar werden. Unternehmen müssen in der Lage sein, ihre vorhandene Software zu ändern und schnell neue Software zu entwickeln. Viele traditionelle Software-Engineering-Techniken sind sehr zeitaufwendig und die Auslieferung neuer Systeme dauert länger als geplant. Diese Techniken müssen weiterentwickelt werden, um die Zeit zu reduzieren, bis die Software den Kunden einen Mehrwert bietet.

3 *Sicherheit und Vertrauen:* Da Software mit allen Aspekten unseres Lebens verflochten ist, müssen wir dieser Software vertrauen können. Dies gilt ganz besonders für entfernte Softwaresysteme, auf die über eine Webseite oder eine Webserviceschnittstelle zugegriffen wird. Wir müssen sicherstellen, dass böswillige Benutzer unsere Software nicht angreifen können und dass die Informationssicherheit gewahrt bleibt.

Natürlich sind diese Themen nicht unabhängig voneinander. Zum Beispiel könnte es erforderlich sein, ein Altsystem schnell anzupassen, um es über eine Webserviceschnittstelle zugänglich zu machen. Um diesen Herausforderungen begegnen zu können, werden wir neue Werkzeuge und Techniken brauchen. Außerdem wird es notwendig sein, auf innovative Art und Weise bestehende Methoden des Software-Engineerings zu kombinieren und anzuwenden.

1.1.2 Vielfalt des Software-Engineerings

Software-Engineering ist ein systematischer Ansatz der Softwareherstellung, der Kosten, Zeitplanung und Verlässlichkeitsprobleme sowie die Bedürfnisse von Softwarekunden und -herstellern berücksichtigt. Wie dieser systematische Ansatz tatsächlich umgesetzt wird, hängt ganz wesentlich von dem Unternehmen ab, das die Software entwickelt, von der Art der Software und von den Menschen, die am Entwicklungsprozess beteiligt sind. Es gibt keine universellen Software-Engineering-Methoden und -Techniken, die für alle Systeme und Unternehmen geeignet sind. Stattdessen hat sich während der letzten 50 Jahre eine facettenreiche Menge von Software-Engineering-Methoden und -Werkzeugen herausgebildet.
Der vielleicht bedeutendste Faktor bei der Frage, welche Software-Engineering-Methoden und -Techniken die wichtigsten sind, ist die Art der zu entwickelnden Anwendung. Es gibt viele unterschiedliche Anwendungsarten, dazu zählen:

1. *Eigenständige* (*stand-alone) Anwendungen:* Dies sind Anwendungssysteme, die auf einem lokalen Rechner wie einem PC laufen. Sie besitzen alle nötigen Funktionalitäten und müssen nicht mit einem Netzwerk verbunden sein. Beispiele solcher Anwendungen sind Office-Anwendungen auf einem PC, CAD-Programme, Software zur Fotobearbeitung usw.

2. *Interaktive transaktionsbasierte Anwendungen:* Diese Anwendungen werden auf einem entfernten Computer ausgeführt. Die Benutzer können entweder von ihren eigenen PCs aus oder über Terminals darauf zugreifen. Hierzu gehören sicherlich Webanwendungen wie E-Commerce-Anwendungen, bei denen man mit einem entfernten System verbunden ist, um Waren und Dienstleistungen zu kaufen. Diese Anwendungsklasse enthält auch Geschäftssysteme, wobei der Zugang zu diesen Systemen über einen Webbrowser oder über spezielle Client-Programm- und Cloud-basierte Dienste wie E-Mail und Foto-Sharing erfolgt. Interaktive Anwendungen benötigen häufig einen großen Datenspeicher, auf den bei jeder Transaktion zugegriffen wird und der dabei jeweils aktualisiert wird.

3. *Eingebettete Steuerungssysteme:* Dies sind Softwaresteuerungssysteme, die Hardwaregeräte steuern und verwalten. Zahlenmäßig gibt es wahrscheinlich mehr eingebettete Systeme als irgendeine andere Art von System. Beispiele für eingebettete Systeme sind die Software in einem Mobiltelefon, Software zur Steuerung des Antiblockiersystems im Auto und Mikrowellensoftware zum Steuern des Kochvorgangs.

4. *Stapelverarbeitende* (*batch processing*) *Systeme:* Dies sind Geschäftssysteme, die zur Verarbeitung großen Datenmengen entworfen wurden. Sie bearbeiten viele individuelle Eingaben, um die dazugehörigen Ausgaben zu erzeugen. Beispiele für Stapelverarbeitungssysteme sind Systeme zur Abrechnung von regelmäßigen Zahlungen wie Telefonabrechnungssysteme und Lohnauszahlungssysteme.

5. *Unterhaltungssysteme:* Dies sind Systeme, die in erster Linie für die private Nutzung gedacht sind und die zur Unterhaltung ihrer Nutzer dienen. Die meisten dieser Systeme sind Spiele. Die Qualität der angebotenen Benutzerinteraktion ist das wichtigste Unterscheidungsmerkmal von Unterhaltungssystemen.

6 *Systeme für die Modellierung und Simulation:* Dies sind Systeme, die von Wissenschaftlern und Ingenieuren entwickelt wurden, um physikalische Vorgänge oder Situationen zu modellieren, in denen viele separate, interagierende Objekte auftreten. Diese Modelle sind oft rechenintensiv und benötigen zur Ausführung parallele Systeme mit hoher Performanz.

7 *Datenerfassungssysteme:* Dies sind Systeme, die mithilfe von Sensoren Daten aus ihrer Umgebung sammeln und diese Daten an andere Systeme zur Verarbeitung senden. Die Software muss mit Sensoren interagieren und wird oft in einer lebensfeindlichen Umgebung oder unter extremen Bedingungen installiert wie beispielsweise innerhalb eines Motors oder an einem unzugänglichen Ort.

8 *Systeme von Systemen:* Diese sind Systeme, die aus vielen anderen Softwaresystemen zusammengesetzt sind. Einige davon können allgemeine Softwareprodukte wie ein Tabellenkalkulationsprogramm sein. Andere Systeme in dem Verbund sind eventuell speziell für diese Umgebung geschrieben worden.

Natürlich sind die Grenzen zwischen diesen Systemtypen unscharf. Wenn Sie ein Spiel für ein Mobiltelefon entwickelt, dann müssen Sie dieselben Beschränkungen (Energieversorgung, Zusammenspiel mit der Hardware) beachten wie die Entwickler der Telefonsoftware. Stapelverarbeitungssysteme werden häufig in Verbindung mit webbasierten Systemen eingesetzt. Zum Beispiel könnten in einer Firma die Reisekostenabrechnungen über eine Webanwendung eingereicht werden, die monatliche Auszahlung findet jedoch über eine Stapelverarbeitung statt.

Man benutzt unterschiedliche Software-Engineering-Techniken für jeden Systemtyp, weil die Software ganz unterschiedliche Eigenschaften hat. Beispielsweise ist ein eingebettetes Steuerungssystem in einem Auto sicherheitskritisch und wird bei der Installation in das Fahrzeug ins ROM gebrannt. Daher ist ein Austausch sehr teuer. Solch ein System erfordert eine sehr umfangreiche **Verifikation und Validierung**, um die Wahrscheinlichkeit eines Rückrufs der Autos nach dem Verkauf zum Beheben von Softwareproblemen zu minimieren. Benutzerinteraktion ist minimal (oder existiert nicht), also sollte auch kein Vorgehensmodell gewählt werden, bei dem zunächst ein Prototyp für die Benutzerschnittstellen entwickelt wird.

Für ein webbasiertes System könnte ein Ansatz angemessen sein, der auf **iterativer Entwicklung** und Auslieferung basiert, da das System aus wiederverwendbaren Komponenten zusammengesetzt ist. Solch ein Vorgehen ist hingegen unpraktisch für ein System von Systemen, für das detaillierte Spezifikationen der Systeminteraktionen im Voraus spezifiziert werden müssen, damit jedes System separat entwickelt werden kann.

Dennoch gibt es einige Grundsätze des Software-Engineerings, die auf alle Arten von Softwaresystemen angewandt werden können:

1 Ein Softwaresystem sollte mithilfe eines gelenkten und verständlichen Entwicklungsprozesses erstellt werden. Das Unternehmen, das die Software entwirft, sollte den Entwicklungsprozess planen und klare Vorstellungen davon haben, was hergestellt wird und wann es fertiggestellt sein wird. Natürlich werden unterschiedliche Prozesse für unterschiedliche Softwaretypen benutzt.

2 Verlässlichkeit und Performanz sind für alle Systemarten wichtig. Software sollte sich wie erwartet verhalten, keine Fehler machen und verfügbar sein, wenn sie benötigt wird. Sie sollte intern sicher in ihren Operationen sein und einen wei-

testgehenden sicheren Schutz gegen äußere Angriffe bieten. Das System sollte effizient arbeiten und keine Betriebsmittel verschwenden.

3 Einen hohen Stellenwert haben die Vereinbarung und die Verwaltung der Softwarespezifikationen und -anforderungen (was die Software tun sollte). Man muss wissen, was die verschiedenen Kunden und Nutzer des Systems von diesem erwarten, und mit ihren Erwartungen umgehen können, damit ein brauchbares System innerhalb des Budgets und gemäß Zeitplan ausgeliefert werden kann.

4 Die vorhandenen Betriebsmittel sollten so effizient wie möglich eingesetzt werden. Dies bedeutet, dass man immer wenn es angebracht ist, bereits bestehende Software wiederverwenden sollte anstatt neue Software zu schreiben.

Diese Grundbegriffe von Prozess, Verlässlichkeit, Anforderungen, Verwaltung und Wiederverwendung sind wichtige Themen dieses Buches. Unterschiedliche Methoden gehen auf unterschiedliche Weise mit diesen Begriffen um, aber sie liegen allen professionellen Softwareentwicklungen zugrunde.

Wie Sie sicher bemerkt haben, werden die Bereiche Implementierung und Programmierung nicht von diesen Grundsätzen erfasst. Ich behandle in diesem Buch keine spezifischen Programmiertechniken, weil diese von einem Systemtyp zum anderen stark variieren. Beispielsweise wird eine Skriptsprache wie Ruby für webbasierte Systemprogrammierung eingesetzt, sie wäre aber für eingebettete Systementwicklung völlig ungeeignet.

1.1.3 Software-Engineering und das Web

Die Entwicklung des Word Wide Web hat unser Leben stark beeinflusst. Anfänglich war das Web hauptsächlich ein universell zugänglicher Informationsspeicher und hatte geringe Auswirkungen auf Softwaresysteme. Diese Systeme liefen auf lokalen Computern und waren nur innerhalb eines Unternehmens zugänglich. Um das Jahr 2000 herum begann das Web sich weiterzuentwickeln und den Browsern wurde immer mehr Funktionalität hinzugefügt. Dies bedeutete, dass webbasierte Systeme entwickelt werden konnten, auf die – anstatt über eine speziell zugeschnittene Benutzerschnittstelle – über einen Webbrowser zugegriffen wurde. Das führte zur Entwicklung einer breiten Palette neuer Systemprodukte, die innovative Dienste lieferten und auf die über das Web zugegriffen wurde. Diese Dienste werden oft durch Werbung finanziert, die auf dem Bildschirm des Nutzers angezeigt wird; damit erübrigt sich ein direktes Bezahlen durch den Nutzer.

Auch die Entwicklung von Webbrowsern, die kleine Programme laufen lassen und einige Funktionen lokal ausführen können, führte zu einer Weiterentwicklung in der Geschäfts- und Unternehmenssoftware. Anstatt Software zu schreiben und sie auf den PCs der Benutzer zu installieren, wurde die Software auf einem Webserver eingerichtet. Dadurch wurde es viel billiger, die Software zu verändern und zu aktualisieren, denn es war nicht nötig, sie auf jedem PC zu installieren. Auch die Gesamtkosten wurden reduziert, da die Entwicklung von Benutzerschnittstellen besonders teuer ist. Folglich haben viele Unternehmen ihre betriebseigenen Softwaresysteme überall, wo dies möglich war, auf webbasierte Interaktion verlegt.

Die nächste Stufe in der Entwicklung von webbasierten Systemen war die Einführung von Webservices. Webservices sind Softwarekomponenten, die bestimmte nützliche Funktionen bieten und auf die über das Web zugegriffen wird. Es wurden Anwendun-

gen entwickelt, die diese Webservices integrieren, die eventuell von verschiedenen Firmen zur Verfügung gestellt wurden. Im Prinzip kann diese Verbindung dynamisch sein, sodass eine Anwendung bei jeder Ausführung unterschiedliche Webserivces in Anspruch nehmen kann. Ich behandle diesen Ansatz zur Softwareentwicklung in *Kapitel 19*.

In den letzten Jahren ist der Begriff „Software as a Service“ (Software als Dienstleistung) entstanden. Dahinter steht die Idee, Software nicht mehr standardmäßig auf lokalen Rechnern ablaufen zu lassen, sondern auf „Computing-Clouds“ (Rechnerwolke), auf die über das Internet zugegriffen wird. Wenn man einen Dienst wie webbasierte E-Mail verwendet, dann benutzt man ein Cloud-basiertes System. Eine Computing-Cloud ist eine riesige Anzahl von miteinander verbundenen Rechnern, die von vielen Anwendern gemeinsam benutzt werden. Die Benutzer kaufen die Software nicht, sondern bezahlen einen Betrag, der davon abhängt, wie oft sie die Software tatsächlich nutzen. Eine andere Möglichkeit ist es, Anwendern freien Zugriff auf die Software als Gegenleistung dafür zu gewähren, dass sie sich Werbung auf ihren Bildschirmen anzeigen lassen.

Das Aufkommen des Webs hat somit zu einer wesentlichen Veränderung in der Gestaltung der Unternehmenssoftware geführt. In der Zeit vor dem Web waren Geschäftsanwendungen überwiegend monolithisch, einzelne Programme liefen auf einzelnen Computern oder Computerclustern. Die Kommunikation fand nur lokal, innerhalb eines Unternehmens statt. Heute ist Software hochgradig verteilt, manchmal über die ganze Welt. Geschäftsanwendungen werden nicht von Grund auf neu programmiert, sondern verwenden ausgiebig bestehende Komponenten und Programme.

Dieser radikale Wandel in der Softwareorganisation hat selbstverständlich zu Veränderungen in der Art und Weise geführt, wie webbasierte Systeme aufgebaut werden. Zum Beispiel:

1. Der vorherrschende Ansatz bei der Konstruktion von webbasierten Systemen ist die Wiederverwendung von Software. Bei der Konstruktion von webbasierten Systemen wird darüber nachgedacht, wie man sie aus bereits vorhandener Software zusammenbauen kann.

2. Es ist heute eine allgemein anerkannte Tatsache, dass es unpraktikabel ist, alle Anforderungen für solch ein System im Voraus zu spezifizieren. Webbasierte Systeme sollten inkrementell entwickelt und ausgeliefert werden.

3. Benutzerschnittstellen werden durch die Leistungsfähigkeit der Webbrowser eingeschränkt. Obwohl Technologien wie AJAX (Holdener, 2008) die Erzeugung einer umfangreichen Benutzeroberfläche innerhalb eines Webbrowsers ermöglichen, sind diese Technologien immer noch schwierig zu benutzen. Häufiger werden Webformulare mit lokalen Skripts eingesetzt. Anwendungsschnittstellen auf webbasierten Systemen sind oft schwächer als die speziell entworfenen Benutzerschnittstellen bei Produkten für PC-Systeme.

Die grundlegenden Ideen des Software-Engineerings, die im vorhergehenden Abschnitt besprochen wurden, lassen sich auf webbasierte Software ebenso anwenden wie auf andere Arten von Softwaresystemen. Die Erfahrung, die bei der Entwicklung von großen Softwaresystemen im 20. Jahrhundert gewonnen wurde, ist auch für webbasierte Software relevant.

1.2 Ethik des Software-Engineerings

Wie bei anderen technischen Disziplinen auch, wird Software-Engineering in einem sozialen und rechtlichen Rahmen ausgeführt, der die Freiheit der Leute einschränkt, die in diesem Bereich arbeiten. Als Softwareentwickler müssen Sie akzeptieren, dass Ihre Arbeit eine größere Verantwortung umfasst als die reine Anwendung technischer Fähigkeiten. Sie müssen sich ethisch und moralisch verantwortungsbewusst verhalten, wenn Sie als professioneller Softwareentwickler respektiert werden wollen.
Es muss nicht extra erwähnt werden, dass die üblichen Regeln von Ehrlichkeit und Integrität eingehalten werden sollten. Sie sollten Ihr Geschick und Ihr Können nicht einsetzen, um sich unehrlich oder auf eine Weise zu verhalten, die die Softwareentwickler als Berufsgruppe in Misskredit bringt. Es gibt jedoch Gebiete, auf denen das Verhalten nicht von Gesetzen geregelt wird, sondern von dem wesentlich abstrakteren Begriff der beruflichen Verantwortung. Dazu gehören:

1. *Vertraulichkeit:* Sie sollten die Vertraulichkeit gegenüber ihren Arbeitgebern oder ihren Kunden bewahren, unabhängig davon, ob eine formelle Vertraulichkeitsvereinbarung unterzeichnet wurde oder nicht.

2. *Kompetenz:* Sie sollten Ihre Kompetenz nicht falsch darstellen. Sie sollten wissentlich keine Aufträge annehmen, die Ihre Kompetenz übersteigen.

3. *Schutz des geistigen Eigentums:* Sie sollten sich mit den örtlich geltenden Gesetzen zum geistigen Eigentum auskennen, wie Patenten und Urheberrechten. Sie sollten unbedingt sicherstellen, dass das geistige Eigentum Ihrer Arbeitgeber und Ihrer Kunden geschützt wird.

4. *Computermissbrauch:* Sie sollten Ihre technischen Fähigkeiten nicht einsetzen, um die Computer anderer zu missbrauchen. Computermissbrauch umfasst die gesamte Palette von relativ harmlosen Begebenheiten (zum Beispiel Computerspiele auf Firmencomputern) bis hin zu extrem ernsten Verstößen (Verbreitung von Viren oder anderer Malware).

Beim Aufstellen ethischer Standards spielen Berufsvereinigungen und Institutionen eine wichtige Rolle. Organisationen wie die ACM (Association for Computing Machinery) und das IEEE (Institute of Electrical and Electronic Engineers) in den USA, die GI (Gesellschaft für Informatik) und der VDI (Verein Deutscher Ingenieure) in Deutschland sowie die British Computer Society veröffentlichen einen **Kodex für professionelles Verhalten** oder ethische Verhaltensregeln (GI, 1997; VDI, 2000). Die Mitglieder dieser Organisationen verpflichten sich, diesen Leitlinien zu folgen, wenn sie die Mitgliedschaft beantragen. Die Verhaltensregeln beschäftigen sich mit grundlegendem ethischen Verhalten im Allgemeinen.
Fachverbände, insbesondere die ACM und das IEEE, haben kooperiert, um einen gemeinsamen ethischen Kodex zu entwerfen. Dieser Kodex besteht sowohl in einer verkürzten Form, die in ▶ Abbildung 1.3 gezeigt wird, als auch in einer längeren Form (Gotterbarn et al., 1999), die die verkürzte Darstellung um Details und Inhalt ergänzt. Die Intention dieses Kodex wird in den ersten beiden Absätzen der langen Form in folgender Weise zusammengefasst:

> *Computer spielen eine zentrale und wachsende Rolle in Handel, Industrie, Verwaltung, Medizin, Bildung, Unterhaltung und der Gesellschaft im Allgemeinen. Softwareentwickler sind diejenigen, die durch direkte Beteiligung oder als*

Dozenten zu Analyse, Spezifikation, Entwurf, Entwicklung, Zertifizierung, Wartung und Tests von Softwaresystemen beitragen. Aufgrund ihrer Rolle bei der Entwicklung von Softwaresystemen haben Softwareentwickler Gelegenheit, Gutes zu tun oder Schaden zu verursachen, andere in die Lage zu versetzen, Gutes zu tun oder Schaden zu verursachen oder andere dahin gehend zu beeinflussen, Gutes zu tun oder Schaden zu verursachen. Um so sicher wie möglich zu sein, dass ihre Anstrengungen benutzt werden, um Gutes zu tun, müssen Softwareentwickler sich verpflichten, Software-Engineering zu einem nützlichen und geachteten Berufsbild zu machen. In Übereinstimmung mit dieser Verpflichtung sollten sich Softwareentwickler an den folgenden ethischen Kodex halten.

Der Kodex enthält acht Prinzipien, die mit dem Verhalten und den Entscheidungen professioneller Softwareentwickler zusammenhängen, zu denen Praktiker, Ausbilder, Manager, Aufsichtführende und Herausgeber von Richtlinien sowie Auszubildende und Studierende dieses Berufs zu zählen sind. Diese Prinzipien beschreiben die ethisch verantwortlichen Beziehungen, die einzelne Personen, Gruppen und Organisationen miteinander eingehen, und die Hauptverpflichtungen, die aus diesen Beziehungen entstehen. Die Klauseln jedes Prinzips illustrieren einige der Verpflichtungen, die mit diesen Beziehungen verbunden sind. Diese Verpflichtungen gründen sich auf die Menschlichkeit der Softwareentwickler, wobei sie den Menschen, die durch ihre Arbeit und die besonderen Elemente des Software-Engineerings einbezogen werden, besondere Aufmerksamkeit schulden. Der Kodex schreibt diese Verpflichtungen als bindend für jeden vor, der sich als Softwareentwickler versteht oder danach strebt, ein Softwareentwickler zu werden.

Ethischer Kodex und professionelles Verhalten des Software-Engineerings

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

Präambel

Diese verkürzte Version der Verhaltensregeln fasst die Bestrebungen auf einer hohen Abstraktionsebene zusammen. Die vollständige Version dieser Erklärung gibt Beispiele und erläutert im Detail, wie diese Bestrebungen unser Handeln als professionelle Softwareentwickler verändert. Ohne diese Bestrebungen können die Details spitzfindig und langwierig werden; ohne die Details können die Aussagen vielversprechend klingen, aber doch inhaltslos bleiben. Zusammen bilden die Bestrebungen und die Details ein zusammenhängendes Regelwerk.

Softwareentwickler sollen sich verpflichten, Analyse, Spezifikation, Entwurf, Entwicklung, Test und Wartung von Software zu einem nützlichen und geachteten Beruf zu machen. In Übereinstimmung mit ihren Verpflichtungen gegenüber der Gesundheit, Sicherheit und dem Wohlergehen der Öffentlichkeit sollen Softwareentwickler sich an die folgenden acht Prinzipien halten:

1. ÖFFENTLICHKEIT – Softwareentwickler sollen in Übereinstimmung mit dem öffentlichen Interesse handeln.
2. KUNDE UND ARBEITGEBER – Softwareentwickler sollen auf eine Weise handeln, die im Interesse ihrer Kunden und ihres Arbeitgebers ist und sich mit dem öffentlichen Interesse deckt.
3. PRODUKT – Softwareentwickler sollen sicherstellen, dass ihre Produkte und damit zusammenhängende Modifikationen den höchstmöglichen professionellen Standards entsprechen.
4. BEURTEILUNG – Softwareentwickler sollen bei der Beurteilung eines Sachverhalts Integrität und Unabhängigkeit wahren.

Abbildung 1.3: Ethischer Kodex von ACM/IEEE (©IEEE/ACM 1999).

5. MANAGEMENT – Für das Software-Engineering verantwortliche Manager und Projektleiter sollen sich bei ihrer Tätigkeit ethischen Grundsätzen verpflichtet fühlen und in diesem Sinne handeln.
6. BERUF – Softwareentwickler sollen die Integrität und den Ruf des Berufs in Übereinstimmung mit dem öffentlichen Interesse fördern.
7. KOLLEGEN – Softwareentwickler sollen sich ihren Kollegen gegenüber fair und hilfsbereit verhalten.
8. SELBST – Softwareentwickler sollen sich einem lebenslangen Lernprozess in Bezug auf ihren Beruf unterwerfen und anderen eine ethische Ausübung des Berufes vorleben.

Abbildung 1.3: Ethischer Kodex von ACM/IEEE (©IEEE/ACM 1999). *(Forts.)*

In jeder Situation, in der verschiedene Personen unterschiedliche Ansichten und Ziele haben, ist es wahrscheinlich, dass Sie sich einem ethischen Dilemma gegenübersehen. Wenn Sie beispielsweise der Firmenpolitik des oberen Managements nicht zustimmen, wie sollen Sie dann reagieren? Das hängt sicherlich von den betroffenen Einzelpersonen und der Art der Unstimmigkeit ab. Ist es besser, innerhalb der Firma für Ihre Ansichten zu argumentieren, oder sollten Sie aus prinzipiellen Überlegungen kündigen? Wenn Sie das Gefühl haben, dass es ein Problem mit einem Softwareprojekt gibt, wann melden Sie dies dem Management? Wenn Sie über die Probleme diskutieren, solange sie nicht mehr als ein Verdacht sind, könnten Sie in dieser Situation überreagieren; wenn Sie es zu lange verschieben, könnte es hingegen zu spät sein, die Probleme zu lösen.

Solchen ethischen Dilemmata stehen wir alle in unserem Berufsleben gegenüber und zum Glück sind sie meist unbedeutend oder können ohne große Schwierigkeiten gelöst werden. Wenn sie nicht gelöst werden können, stehen Entwickler vielleicht vor einem neuen Problem. Aus prinzipiellen Gründen sollten sie ihren Job vielleicht kündigen, aber das ist eine Entscheidung, die auch andere betrifft, wie zum Beispiel die Partner oder die Kinder.

Eine für einen Entwickler besonders schwierige Situation entsteht, wenn der Arbeitgeber sich auf unethische Art und Weise verhält. Dies wäre zum Beispiel der Fall, wenn eine Firma für die Entwicklung eines sicherheitskritischen Systems verantwortlich ist und unter Zeitdruck die Berichte aus der Sicherheitsvalidierung fälscht. Ist der Entwickler verpflichtet, sich an die Vertraulichkeit zu halten, oder muss er den Kunden benachrichtigen oder auf die eine oder andere Weise bekannt machen, dass das betreffende System unsicher sein könnte?

Das Problem besteht in dieser Situation darin, dass es keine absoluten Maßstäbe für Sicherheit gibt. Selbst wenn das System nicht gemäß vordefinierter Kriterien validiert wurde, könnten diese Kriterien auch zu streng sein. Das System könnte tatsächlich ein Leben lang sicher arbeiten. Genauso wäre es möglich, dass das System zusammenbricht, obwohl es korrekt validiert wurde. Ein frühes Aufdecken von Problemen kann den Arbeitgeber und andere Mitarbeiter schädigen; das Verschweigen von Problemen kann andere in Mitleidenschaft ziehen.

Sie müssen sich zu diesen Fragen Ihre eigene Meinung bilden. Der angemessene ethische Standpunkt hängt ganz von den jeweils handelnden Personen und ihren Überzeugungen ab. In diesem Fall sollten das Schadenspotenzial, das Ausmaß des Schadens und die Menschen, die dieser Schaden betrifft, die Entscheidung beeinflussen. Wenn die Situation sehr ernst ist, kann es gerechtfertigt sein, zum Beispiel die Presse zu verständigen. Sie sollten jedoch immer versuchen, eine Lösung zu finden, die auch die Rechte Ihres Arbeitgebers schützt.

Ein anderes ethisches Problem ist die Entwicklung von militärischen und nuklearen Systemen. Einige Menschen sind strikt dagegen und wollen an keiner Systementwicklung teilnehmen, die mit militärischen Systemen zusammenhängt. Andere sind bereit, an militärischen Systemen zu arbeiten, aber nicht an Waffensystemen. Wieder andere glauben, dass die nationale Verteidigung ein höheres Gut ist, und haben somit keine ethischen Bedenken, an Waffensystemen mitzuarbeiten.

In dieser Situation ist es wichtig, dass sowohl Arbeitgeber als auch Arbeitnehmer sich schon im Vorfeld gegenseitig über ihre jeweiligen Standpunkte aufklären. Eine Organisation, die mit militärischen oder nuklearen Projekten zu tun hat, sollte verlangen können, dass die Arbeitnehmer alle möglichen Aufgaben akzeptieren. Genauso sollten Arbeitnehmer, die eingestellt wurden und klargestellt haben, dass sie nicht an solchen Systemen arbeiten wollen, zu keinem späteren Zeitpunkt von ihrem Arbeitgeber deswegen unter Druck gesetzt werden.

Das allgemeine Gebiet der Ethik und der professionellen Verantwortung wird immer wichtiger, da softwareintensive Systeme jeden Aspekt der Arbeit und des alltäglichen Lebens durchdringen. Es kann auch von einem philosophischen Standpunkt betrachtet werden, der die Grundprinzipien der Ethik heranzieht und die Ethik des Software-Engineerings im Hinblick auf diese Grundprinzipien diskutiert. Das ist der Ansatz, den Laudon (1995) und in abgeschwächter Form auch Huff und Martin (1995) verfolgen. Auch der Text von Johnson (2001) über Computerethik nähert sich diesem Thema aus einer philosophischen Perspektive.

Ich halte diesen philosophischen Ansatz jedoch für zu abstrakt und denke, dass er nur schwer mit meinen Erfahrungen im Alltag zu vereinbaren ist. Ich bevorzuge den konkreteren Ansatz, dem man in Verhaltensregeln Ausdruck verleiht. Ich glaube, dass Ethik am besten im Zusammenhang mit Software-Engineering diskutiert wird und nicht als eigenes Thema. Daher enthält dieses Buch keine abstrakten ethischen Erörterungen, sondern gelegentlich Beispiele in den Übungen, die als Ausgangspunkt für ethische Diskussionen innerhalb einer Gruppe dienen können.

1.3 Fallstudien

Um die Konzepte des Software-Engineerings zu illustrieren, verwende ich im gesamten Buch Beispiele von drei unterschiedlichen Systemtypen. Ich benutze nicht nur eine einzelne Fallstudie, weil eine der Kernaussagen dieses Buches ist, dass die Verfahren im Software-Engineering von den herzustellenden Systemtypen abhängen. Ich wähle deshalb jeweils ein passendes Beispiel bei der Besprechung von Konzepten wie Betriebssicherheit und Verlässlichkeit, Systemmodellierung, Wiederverwendung usw.

Die drei Systemarten, die ich als Fallstudien benutze, sind:

1. *Ein eingebettetes System:* Dies ist ein System, bei dem Software ein Hardwaregerät steuert und in diesem Gerät eingebettet ist. Kernfragen bei eingebetteten Systemen beschäftigen sich typischerweise mit der physischen Größe, Antwortverhalten, Energieverwaltung usw. Das hier gewählte Beispiel eines eingebetteten Systems ist ein Softwaresystem zur Steuerung eines medizinischen Geräts.

2. *Ein Informationssystem:* Dies ist ein System, dessen hauptsächlicher Zweck es ist, Zugang zu einer Datenbank von Informationen zu verwalten und zur Verfügung zu stellen. Kernfragen bei Informationssystemen schließen Informationssicherheit, Benutzerfreundlichkeit, Datenschutz und Erhalt der Datenintegrität

ein. Als Beispiel für ein Informationssystem stelle ich ein Patientendatenmanagementsystem vor.

3 *Ein sensorbasiertes Datenerfassungssystem:* Dies ist ein System, dessen Hauptzweck es ist, Daten von einer Menge von Sensoren zu sammeln und diese Daten in irgendeiner Weise zu verarbeiten. Die Schlüsselanforderungen eines solchen Systems sind Zuverlässigkeit – selbst unter feindlichen Umweltbedingungen – und Wartbarkeit. Das Beispiel eines Datenerfassungssystem, das ich hier verwende, ist eine Wetterstation in Wildnisgebieten.

Ich stelle im Folgenden jedes dieser Systeme kurz vor, weitere Informationen darüber sind im Web verfügbar.

1.3.1 Ein Steuerungssystem für Insulinpumpen

Eine **Insulinpumpe** ist ein medizinisches System, das die Funktion der Bauchspeicheldrüse (einem inneren Organ) nachahmt. Die Softwaresteuerung dieses Systems ist ein eingebettetes System, das Informationen von einem Sensor sammelt und ein Pumpe steuert, die eine kontrollierte Insulindosis an den Benutzer abgibt.

Das System wird von Menschen benutzt, die an Diabetes leiden. Diabetes ist eine häufige Erkrankung, bei der die menschliche Bauchspeicheldrüse nicht in der Lage ist, eine ausreichende Menge des Hormons Insulin zu produzieren. Insulin baut Glukose (Zucker) im Blut ab. Die übliche Behandlung von Diabetes umfasst regelmäßige Injektionen von künstlich hergestelltem Insulin. Diabetiker messen ihren Blutzuckerspiegel, indem sie ein externes Messgerät benutzen und dann die Insulindosis berechnen, die sie injizieren sollen.

Das Problem bei dieser Vorgehensweise ist, dass der benötigte Insulinspiegel nicht nur vom Blutzuckerspiegel, sondern auch von dem Zeitpunkt der letzten Insulininjektion abhängt. Dies kann zu sehr niedrigen Blutzuckerspiegeln führen (falls zu viel Insulin im Blut ist) oder zu sehr hohen Blutzuckerspiegeln (wenn zu wenig Insulin vorhanden ist). Kurzfristig gesehen ist ein niedriger Blutzuckerspiegel ein ernsterer Zustand, da er zu zeitweiligen Gehirnfehlfunktionen und schließlich zur Bewusstlosigkeit und zum Tod führen kann. Auf der anderen Seite kann ein ständig zu hoher Blutzuckerspiegel längerfristig zu Augenschädigungen, Nierenschäden und Herzproblemen führen.

Aktuelle Fortschritte bei der Entwicklung von miniaturisierten Sensoren haben zu der Entwicklung von automatischen Insulinabgabesystemen geführt. Diese Systeme überwachen den Blutzuckerspiegel und geben bei Bedarf eine angemessene Insulindosis ab. Insulinabgabesysteme wie dieses existieren bereits für die Behandlung von Krankenhauspatienten. Zukünftig kann es aber für viele Diabetiker möglich sein, solche Systeme dauerhaft an ihrem Körper zu tragen.

Ein softwarekontrolliertes Insulinabgabesystem könnte mithilfe eines Mikrosensors arbeiten, der im Patienten implantiert ist, um einige Blutparameter zu messen, die proportional zum Zuckerspiegel sind. Diese Werte werden dann zur Pumpensteuerung gesendet. Die Steuerung berechnet den Zuckerspiegel und die benötigte Insulinmenge. Dann sendet sie Signale an eine Miniaturpumpe, um das Insulin über eine permanent angebrachte Nadel abzugeben.

▶ Abbildung 1.4 zeigt die Hardwarekomponenten und -organisation der Insulinpumpe. Um die Beispiele in diesem Buch zu verstehen, müssen Sie lediglich wissen, dass der Blutsensor die elektrische Leitfähigkeit des Bluts unter verschiedenen Bedingungen misst und dass diese Werte zum Blutzuckerspiegel in Bezug gesetzt werden

können. Die Insulinpumpe gibt eine Insulineinheit als Reaktion auf einen einzelnen kurzen Stromstoß der Steuerungseinheit ab. Deshalb sendet die Steuerung zehn Impulse an die Pumpe, um zehn Insulineinheiten abzugeben. ▶ Abbildung 1.5 ist ein UML-Aktivitätsmodell, das illustriert, wie die Software eine Blutzuckerspiegeleingabe in eine Folge von Befehlen umwandelt, die wiederum die Insulinpumpe ansteuern.

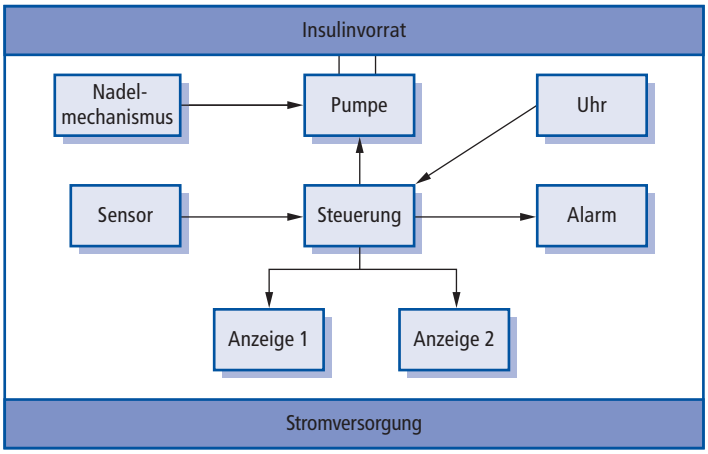


Abbildung 1.4: Hardware der Insulinpumpe.

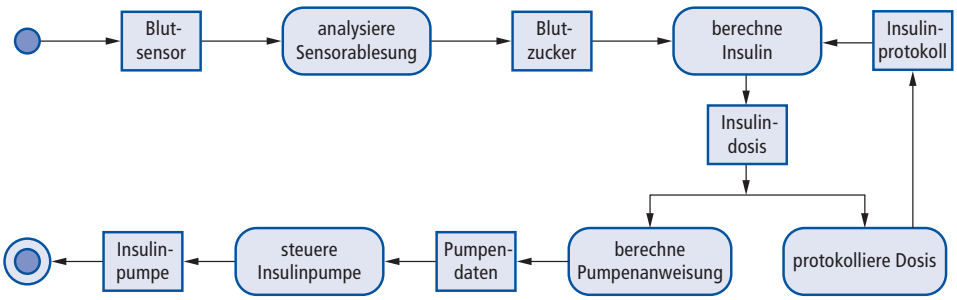


Abbildung 1.5: Aktivitätsmodell der Insulinpumpe.

Dies ist natürlich ein sicherheitskritisches System. Falls die Pumpe ausfällt oder nicht korrekt arbeitet, kann die Gesundheit der Benutzer beeinträchtigt werden oder sie können in ein Koma fallen, weil ihre Blutzuckerspiegel zu hoch oder zu niedrig sind. Es gibt deshalb zwei wesentliche Anforderungen auf höherer Ebene, die dieses System erfüllen muss:

1. Das System soll zur Insulinabgabe bereit sein, wenn diese angefordert wird.
2. Das System soll zuverlässig arbeiten und die richtige Insulinmenge abgeben, um dem aktuellen Blutzuckerspiegel entgegenzuwirken.

Der Entwurf und die Implementierung des Systems müssen daher sicherstellen, dass diese Anforderungen immer erfüllt werden. Detailliertere Anforderungen und Diskussionen darüber, wie die Betriebssicherheit des Systems gewährleistet wird, folgen in späteren Kapiteln.

1.3.2 Ein Patienteninformationssystem für die psychiatrische Ambulanz

Ein Patienteninformationssystem zur Unterstützung einer psychiatrischen Ambulanz ist ein medizinisches Informationssystem, das Informationen über Patienten mit psychischen Problemen und über die durchgeführten Behandlungen verwaltet. Die meisten dieser Patienten benötigen keine stationäre Krankenhausbehandlung, sondern es reicht in der Regel aus, wenn sie regelmäßig in die ambulante Sprechstunde kommen, um einen Arzt zu konsultieren, der genaue Kenntnisse von ihren Problemen hat. Um diesen Patienten den Arztbesuch zu erleichtern, werden diese Sprechstunden nicht nur in Krankenhäusern abgehalten. Sie können auch in lokale Arztpraxen oder Polikliniken abgehalten werden.

Das MHC-PMS (*Mental Health Care-Patient Management System*) ist ein Informationssystem, das zur Verwendung in solchen ambulanten Sprechstunden vorgesehen ist. Das System verwendet eine zentralisierte Datenbank mit Patienteninformationen, es kann aber auch auf einem PC laufen, sodass man darauf zugreifen und das System benutzen kann, auch wenn keine sichere Netzwerkverbindung vorhanden ist. Falls die lokalen Systeme einen sicheren Netzzugang haben, so benutzen sie die Patienteninformation in der Datenbank. Ansonsten können sie lokale Kopien von Patientendatensätzen herunterladen und mit diesen arbeiten, wenn sie nicht im Netz sind. Das System ist kein vollständiges Patientendatenmanagementsystem, es enthält zum Beispiel keine Informationen über sonstige Krankheiten des Patienten. Es könnte jedoch mit anderen klinischen Informationssysteme interagieren und Daten austauschen. ▶ Abbildung 1.6 zeigt den Aufbau des MHC-PMS.

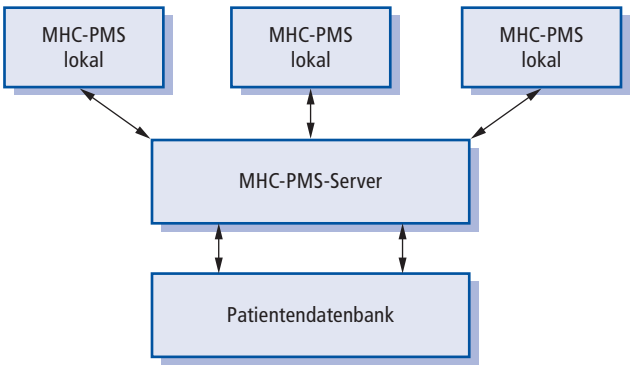


Abbildung 1.6: Die Organisation des MHC-PMS.

Das MHC-PMS hat zwei übergeordnete Ziele:

1. Es soll Verwaltungsinformationen erzeugen, die es Verwaltungsangestellten im Gesundheitswesen erlauben, die Leistung gegenüber lokalen und Regierungszielen zu bewerten.

2. Das medizinische Personal soll zeitnah mit Informationen versorgt werden, um die Behandlung der Patienten zu unterstützen.

Bei psychischen Gesundheitsproblemen kommt es häufig vor, dass Patienten desorganisiert sind und daher Termine verpassen, absichtlich oder unabsichtlich Rezepte und

Medikamente verlieren, Anweisungen vergessen und unangemessene Forderungen an das medizinische Personal stellen. Sie könnten unangemeldet in die Sprechstunde kommen. In einigen wenigen Fällen stellen die Patienten eine Gefahr für sich selbst oder für andere Menschen dar. Es kann passieren, dass sie regelmäßig ihre Adresse ändern oder kurzzeitig oder länger obdachlos sind. Wenn Patienten eine Gefahr darstellen, wird es eventuell notwendig, sie in eine geschlossene psychiatrische Klinik zur Behandlung und Beobachtung einzuweisen.

Benutzer des Systems sind das klinische Personal wie Ärzte, Krankenpfleger und ambulante Gesundheitsdienste (Pfleger, die Patienten zu Hause besuchen, um ihre Behandlung zu überprüfen). Nichtmedizinische Nutzer sind das Empfangspersonal, das die Termine macht, die Mitarbeiter, die das Patientendatensystem pflegen, und Verwaltungsangestellte, die Berichte erzeugen.

Das System wird eingesetzt, um Informationen über Patienten aufzuzeichnen (Name, Adresse, Alter, nächste Angehörige usw.), Konsultationen (Datum, behandelter Arzt, subjektive Eindrücke des Patienten usw.), Krankheitsbilder und Behandlungen. Berichte werden in regelmäßigen Abständen für das medizinische Personal und für die Verwaltungen der Gesundheitsbehörden erstellt. Typischerweise konzentrieren sich die Berichte für das medizinische Personal auf einzelne Patienten, wohingegen Verwaltungsberichte anonymisiert sind und sich mit Bedingungen, Kosten der Behandlung usw. beschäftigen.

Die Schlüsselmerkmale des Systems sind:

1. *Individuelle Patientenbetreuung:* Fachärzte können Datensätze für Patienten erstellen, die vorhandenen Informationen im System bearbeiten, die Krankengeschichte ansehen usw. Das System unterstützt das Zusammenfassen von Daten, sodass sich auch Ärzte, die einen Patienten zuvor noch nicht behandelt haben, schnell über die Hauptprobleme und bisher durchgeführten Maßnahmen informieren können.

2. *Patientenüberwachung:* Das System überwacht regelmäßig die Datensätze der Patienten, die sich in Behandlung befinden, und warnt, wenn mögliche Probleme entdeckt werden. Falls also ein Patient länger nicht zur Sprechstunde gekommen ist, könnte eine Warnmeldung ausgegeben werden. Einer der wichtigsten Elemente des Überwachungssystem ist es, den Überblick über die Patienten zu behalten, die in eine psychiatrische Klinik eingewiesen wurden, und sicherzustellen, dass die gesetzlich vorgeschriebenen Kontrollen zum richtigen Zeitpunkt ausgeführt wurden.

3. *Verwaltungsberichte:* Das System erstellt monatlich Verwaltungsberichte, die die Anzahl der Patienten zeigt, die in jeder Sprechstunde behandelt wurden, die Anzahl der Patienten, die in das Pflegesystem ein- und aus diesem wieder ausgebucht wurden, die Anzahl der Patienten, die in die Psychiatrie eingewiesen wurden, die verschriebenen Arzneimittel und ihre Kosten usw.

Zwei unterschiedliche Gesetze beeinflussen das System. Dies sind zum einen Gesetze zum Datenschutz, die die Vertraulichkeit von persönlichen Informationen regeln, und zum anderen Gesetze zum Umgang mit psychisch Kranken, die die Zwangseinweisung von Patienten regeln, die eine Gefahr für sich oder andere darstellen. Psychische Krankheiten sind in dieser Hinsicht einzigartig, denn es ist das einzige medizinische Spezialgebiet, das die Einweisung von Patienten gegen ihren Willen anordnen kann.

Diese Einweisung unterliegt sehr strengen gesetzlichen Schutzvorschriften. Eines der Ziele von MHC-PMS ist es sicherzustellen, dass das Personal immer im Einklang mit dem Gesetz handelt und dass ihre Entscheidungen für eine eventuelle richterliche Überprüfung aufgezeichnet werden.

Wie in allen medizinischen Systemen ist die Vertraulichkeit der Daten eine kritische Systemforderung. Es ist unbedingt notwendig, dass Patienteninformationen vertraulich behandelt werden und niemals weitergegeben werden an andere Personen außer dem autorisierten medizinischen Personal und dem Patienten selbst. MHC-PHS ist ebenfalls ein sicherheitskritisches System. Einige psychische Krankheiten bringen Patienten dazu, an Selbstmord zu denken oder zu einer Gefahr für andere Menschen zu werden. Wo immer es möglich ist, sollte das System das medizinische Personal über potenziell selbstmordgefährdete oder gefährliche Patienten informieren.

Der Gesamtentwurf des Systems muss Datenschutz- und Betriebssicherheitsanforderungen berücksichtigen. Das System muss bei Bedarf zur Verfügung stehen, ansonsten könnte die Betriebssicherheit gefährdet sein und es könnte unmöglich sein, dem Patienten bei Bedarf die richtigen Medikamente zu verschreiben. Hier zeigt sich ein potenzieller Konflikt: Der Datenschutz ist einfacher zu erhalten, wenn es nur eine einzige Kopie der Systemdaten gibt. Um jedoch die Verfügbarkeit im Fall eines Serverausfalls oder bei Unterbrechung der Netzwerkverbindung zu erhalten, sollten mehrere Kopien der Daten vorgehalten werden. Ich bespreche die Abwägung zwischen diesen Anforderungen in späteren Kapiteln.

1.3.3 Eine Wetterstation in Wildnisgebieten

Um bei der Beobachtung von Klimaveränderungen zu helfen und um die Genauigkeit von Wettervorhersagen in abgelegenen Gebieten zu verbessern, entscheidet sich die Regierung eines Landes mit großen Wildnisflächen, mehrere Hundert Wetterstationen in abgelegenen Gebieten einzurichten. Diese Wetterstationen sammeln Daten von einer Reihe von Instrumenten, die Temperatur und Druck, Sonnenschein, Niederschlag, Windgeschwindigkeit und Windrichtung messen.

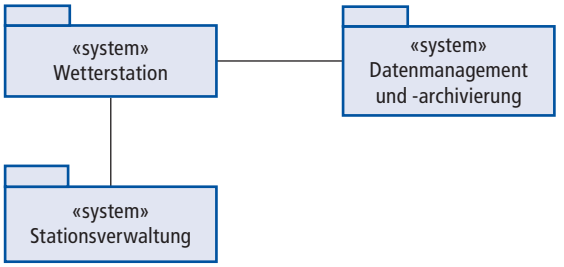


Abbildung 1.7: Die Umgebung der Wetterstationen.

Die Wildnis-Wetterstationen sind Teil eines größeren Systems (▶ Abbildung 1.7) – ein Wetterinformationssystem, das Daten von Wetterstationen sammelt und diese anderen Systemen zur Verarbeitung zugänglich macht. Die Systeme in Abbildung 1.7 sind:

1. *Das Wetterstationssystem:* Dies ist verantwortlich für das Sammeln der Wetterdaten, für die Ausführung einiger initialer Datenverarbeitungen und für die Übertragung dieser Daten zum Datenmanagementsystem.

2. *Das Datenmanagement- und Archivierungssystem:* Dieses System sammelt die Daten von allen Wildnis-Wetterstationen, verarbeitet Daten und analysiert sie und archiviert die Daten in einer Form, die von anderen Systemen abgerufen werden kann, wie beispielsweise Wettervorhersagesystemen.

3. *Das Stationsverwaltungssystem:* Dieses System kann über Satellit mit allen Wildnis-Wetterstationen kommunizieren und so die Funktionstüchtigkeit dieser Systeme überwachen und Berichte über Probleme liefern. Es kann die eingebettete Software in diesen Systemen aktualisieren. Im Falle eines Systemproblems kann dieses System auch benutzt werden, um ein Wildnis-Wettersystem aus der Ferne zu steuern.

In Abbildung 1.7 habe ich das UML-Paketsymbol benutzt um anzuzeigen, dass jedes System eine Sammlung von Komponenten ist. Zur Kennzeichnung der einzelnen Systeme habe ich das UML-Stereotype «system» verwendet. Die Assoziationen zwischen den Paketen zeigen, dass dort ein Informationsaustausch stattfindet, doch im Moment ist es nicht nötig, diesen genauer zu definieren.

Jede Wetterstation enthält eine Anzahl von Instrumenten, die Wetterparameter wie Windstärke und -richtung, Boden- und Lufttemperatur, Barometerdruck und Niederschlag innerhalb von 24 Stunden messen. Jedes dieser Instrumente wird von einem Softwaresystem gesteuert, das periodisch Parameterablesungen vornimmt und die Daten verwaltet, die von den Instrumenten gesammelt werden.

Das Wetterstationssystem sammelt Wetterbeobachtungen in kurzen Abständen – Temperaturen werden beispielsweise jede Minute gemessen. Da jedoch die Bandbreite zum Satellit relativ schmal ist, werden einige Daten von der Wetterstation lokal verarbeitet und zusammengefasst. Diese aggregierten Daten werden dann übertragen, wenn das Datenerfassungssystem sie anfordert. Falls es – warum auch immer – unmöglich ist, eine Verbindung herzustellen, dann pflegt die Wetterstation die Daten lokal, bis der Austausch fortgesetzt werden kann.

Jede Wetterstation ist batteriebetrieben und muss vollständig unabhängig sein – es gibt keine externe Energieversorgung oder Netzwerkkabel. Die gesamte Kommunikation läuft über eine relativ langsame Satellitenverbindung und die Wetterstation muss einige Mechanismen (Solar- oder Windenergie) vorhalten, um ihre Batterien wieder aufzuladen. Da die Stationen in Wildnisgebieten eingesetzt werden, sind sie harten Umweltbedingungen ausgesetzt und könnten von Wildtieren beschädigt werden. Die Stationssoftware ist daher nicht nur mit der Datenerfassung befasst. Sie muss ebenso

1. die Instrumente, Energieversorgung und Kommunikationshardware überwachen und Fehler an das Managementsystem melden;

2. die Systemenergie verwalten um sicherzustellen, dass zum einen die Batterien aufgeladen werden, wenn die Umweltbedingungen es zulassen, und dass zum anderen die Generatoren bei potenziell schädlichen Wetterverhältnissen wie beispielsweise starkem Wind abgeschaltet werden;

3. die dynamische Rekonfigurierung erlauben, wobei Teile der Software mit neuen Versionen ersetzt werden und im Fall eines Systemausfalls auf Backup-Instrumente umgeschaltet wird.

Weil Wetterstationen unabhängig und unbeaufsichtigt funktionieren müssen, ist die installierte Software recht komplex, selbst wenn die Tätigkeit des Datensammelns vergleichsweise einfach ist.

Zusammenfassung

- Software-Engineering ist eine Fachdisziplin, die sich mit allen Aspekten der Softwareherstellung beschäftigt.
- Software besteht nicht nur aus einem oder mehreren Programmen, sondern umfasst auch die Dokumentation. Wesentliche Eigenschaften von Softwareprodukten sind Wartbarkeit, Verlässlichkeit, Informationssicherheit, Effizienz und Benutzerfreundlichkeit.
- Der Softwareprozess umfasst alle Aktivitäten, die mit der Entwicklung von Software zusammenhängen. Grundlegende Aktivitäten sind Spezifikation, Entwicklung, Validierung und Evolution.
- Die grundlegenden Notationen des Software-Engineerings sind universell auf alle Typen der Softwareentwicklung anwendbar. Diese Grundlagen beinhalten Softwareprozesse, Verlässlichkeit, Informationssicherheit, Anforderungen und Wiederverwendung.
- Es gibt viele verschiedene Systemarten und jede benötigt entsprechende Software-Engineering-Werkzeuge und -Techniken für ihre Entwicklung. Es gibt wenige – falls überhaupt – spezifische Entwurfs- und Implementierungstechniken, die auf alle Systemarten anwendbar sind.
- Die grundlegenden Konzepte des Software-Engineerings sind auf alle Typen von Softwaresystemen anwendbar. Diese Grundlagen beinhalten geführte Softwareprozesse, Softwareverlässlichkeit und Informationssicherheit, Requirements-Engineering und Wiederverwendung von Software.
- Softwareentwickler haben eine Verantwortung gegenüber ihrem Beruf und der Gesellschaft. Sie sollten sich nicht nur mit technischen Problemen beschäftigen.
- Berufsverbände veröffentlichen Verhaltensregeln, die die Standards für das Verhalten ihrer Mitglieder festlegen.

Ergänzende Literatur

„No silver bullet: Essence and accidents of software engineering". Trotz seines Alters gibt dieser Artikel eine gute allgemeine Einführung in die Probleme des Software-Engineerings. Die wesentliche Botschaft des Artikels hat sich noch nicht geändert. (F.P. Brooks, *IEEE Computer*, 20(4), April 1987.) *http://doi.ieeecomputersociety.org/10.1109/MC.1987.1663532.*

„Software-Engineering Code of Ethics is approved". Ein Artikel, der den Hintergrund der Entwicklung des Ethikkodex von ACM/IEEE betrachtet und der sowohl die lange als auch die verkürzte Version der Leitlinien enthält. (D. Gotterbarn, K. Miller und S. Rogerson, *Comm. ACM*, Oktober 1999.) *http://portal.acm.org/citation.cfm?doid=317665.317682.*

Professional Issues in Software Engineering. Ein exzellentes Buch über rechtliche und berufliche Probleme sowie über Ethik. Ich bevorzuge den hier gezeigten praktischen Ansatz gegenüber den mehr theoretischen Abhandlungen zur Ethik. (F. Bott, A. Coleman, J. Eaton und D. Rowland, 3. Auflage, Taylor & Francis, 2000.)

IEEE Software, March/April 2002. Eine spezielle Ausgabe des Magazins, das der Entwicklung von webbasierter Software gewidmet ist. Dieser Bereich hat sich sehr schnell verändert, sodass einige Artikel ein wenig überholt sind, doch die meisten sind noch relevant. (*IEEE Software*, 19(2), 2002.) *http://www2.comuter.org/portal/web/software.*

„A View of 20th and 21st Century Software Engineering". Ein Vorschau und Rückblick auf Software-Engineering von einem der ersten und hervorragendsten Softwareentwickler. Barry Boehm identifiziert zeitlose Prinzipien des Software-Engineerings, aber zeigt auch auf, dass einige der gemeinhin angewendeten Vorgehensweisen überflüssig sind. (B. Boehm, *Proc. 28th Software Engineering Conf.*, Shanghai, 2006.)

http://doi.ieeecomputersociety.org/10.1145/1134285.1134288.

„Software Engineering Ethics". Spezielle Ausgabe von IEEE Computer mit vielen Artikeln zum Thema. (*IEEE Computer*, 42(6), Juni 2009.)

Übungen

1 Erklären Sie, warum professionelle Software nicht nur die Programme umfasst, die für einen Kunden entwickelt werden.

2 Was ist der wichtigste Unterschied zwischen generischer Softwareproduktentwicklung und kundenspezifischer Softwareentwicklung? Was könnte das in der Praxis für Benutzer von generischen Softwareprodukten bedeuten?

3 Welches sind die vier wichtigsten Merkmale, die alle professionellen Softwareprodukte aufweisen sollten? Schlagen Sie vier andere Merkmale vor, die gegebenenfalls auch wichtig sein könnten.

4 Zeigen Sie, abgesehen von den Herausforderungen der Heterogenität, des unternehmerischen und sozialen Wandels, des Vertrauens und der Sicherheit, andere Probleme und Herausforderungen auf, denen Software-Engineering im 21. Jahrhundert wahrscheinlich begegnen wird. (Hinweis: Denken Sie an die Umwelt.)

5 Erklären Sie mit Beispielen auf der Grundlage Ihrer eigenen Kenntnisse von einigen der Anwendungstypen, die in Abschnitt 1.1.2 besprochen wurden, warum unterschiedliche Anwendungstypen spezialisierte Software-Engineering-Techniken erforderlich machen, um deren Entwurf und Entwicklung zu unterstützen.

6 Erklären Sie, warum es grundlegende Konzepte des Software-Engineerings gibt, die auf alle Typen von Softwaresystemen angewandt werden.

7 Erklären Sie, wie die universelle Benutzung des Webs Softwaresysteme verändert hat.

8 Diskutieren Sie, ob professionelle Entwickler ebenso wie Ärzte und Anwälte zertifiziert sein sollten.

9 Schlagen Sie für jeden Punkt der ethischen Regeln von ACM/IEEE in Abbildung 1.3 ein passendes Beispiel vor, das den entsprechenden Punkt veranschaulicht.

10 Zur Terrorbekämpfung planen zahlreiche Länder die Entwicklung von Computersystemen (oder haben diese bereits entwickelt), die Bürger und deren Aktivitäten überwachen sollen. Dies hat eindeutig Auswirkungen auf den Datenschutz. Erläutern Sie die ethische Überlegungen, die man anstellen könnte, wenn man an der Entwicklung eines solchen Systems beteiligt ist.