# 1 Allgemeine Aspekte

Ein neues Programm zu schreiben und nach erfolgreicher Übersetzung auf einem Rechner zum Leben zu erwecken gehört sicher zu den spannenden Momenten im beruflichen Alltag des Softwareentwicklers. Entwurfsideen präzise zu formulieren oder Programme ausreichend zu dokumentieren wird hingegen eher zu den langweiligen »Muss-Aufgaben« gezählt. Zur Motivation der Leserschaft, sich trotz dieser verständlichen Tendenz zur Implementierung mit allen Aspekten der methodischen Softwareentwicklung zu befassen, resümiert Abschnitt 1.1 die Softwareentwicklung im Laufe der letzten Jahrzehnte [Sneed99]. Abschnitt 1.2 charakterisiert dann den aktuellen Stand.

## 1.1 Softwareentwicklung gestern

Softwareentwicklung war schon immer ein risikoreiches Geschäft mit einer hohen Misserfolgsquote. In den 70er Jahren belegten verschiedene Studien, dass ungefähr jedes dritte Projekt zum Scheitern verurteilt war. Aber auch bei den so genannten »erfolgreichen« Projekten waren Zeit- und Kostenüberschreitungen an der Tagesordnung. Verschiedene damalige Studien kamen zu dem Schluss, dass ca. 85 % der Projekte ihre geplanten Kosten um mehr als 15 % überschritten. Aber das war die Softwareentwicklung in den 70er Jahren, die dadurch gekennzeichnet war, dass es keine Prozessmodelle, keine Schätzmethoden, kein Qualitätsmanagement und keine Objektorientierung gab. Zudem steckte die Disziplin des Software Engineering noch in den Kinderschuhen und unterstützende Werkzeuge mit grafischen Oberflächen, die auf einem persönlichen Arbeitsplatz liefen, gab es damals noch nicht. Eigentlich könnte man deshalb davon ausgehen, dass die Softwareentwicklung sich heutzutage erfolgreicher darstellen müsste.

Eine Untersuchung der Standish-Gruppe [Standish95] im Jahre 1994 wollte die Fortschritte der amerikanischen Softwarebranche mit aussagekräftigen Zahlen belegen. Die Ergebnisse waren ernüchternd. Es wurden 369 Firmen und 8.380 Projekte berücksichtigt. Das durchschnittliche Projekt in Großbetrieben kostete 2.322.000 Dollar, in Mittelbetrieben 1.331.000 Dollar und in Kleinbetrieben immerhin noch 434.000 Dollar. Von den untersuchten Projekten wurden 31 % noch vor der Fertigstellung aufgegeben, jedes zweite Projekt überschritt sein Budget um mehr als 100 % und jedes dritte Projekt konnte seine ursprünglich gesetzten Ziele nie erreichen. Lediglich 16 % der Projekte wurden im vorgegebenen Zeit- und Budgetrahmen abgeschlossen und konnten somit als »erfolgreich« bewertet werden.

Die Studie belegt auch den negativen Zusammenhang zwischen Projektgröße und Projekterfolg. So beträgt die Wahrscheinlichkeit, dass große Projekte (Projektteam größer 20 Personen) erfolgreich beendet werden, nur ca. 42 %. Bei Projekten mit einer Projektgröße von 5 bis 20 Personen liegt die Erfolgswahrscheinlichkeit bei 65 % und bei Projekten mit weniger als 5 Beteiligten besteht eine 74 %ige Chance.

Als besonders spektakuläre Misserfolge in den USA wurden in der Studie u. a. die folgenden Projekte zitiert:

- das PKW-Zulassungsprojekt des Staates Kalifornien mit einem Verlust von 54 Millionen US-Dollar nach 6 Jahren Entwicklung,
- das American Airlines Autovermietungs- und Hotelvermittlungsprojekt mit 165 Millionen US-Dollar Verlust nach 7 Jahren Entwicklung,
- das Gepäckverteilungssystem des Flughafens Denver, das über Jahre nicht fertig gestellt werden konnte und während dieser Zeit der Stadt Denver einen täglichen Verlust von 1,1 Millionen US-Dollar bescherte und
- das zentrale Steuerverwaltungssystem des U.S. Internal Revenue Service, dessen Entwicklung schon über 8 Jahre dauert, mehr als 200 Millionen US-Dollar gekostet hat und immer noch nicht einsatzfähig ist.

Andere spektakuläre Misserfolge werden aus England gemeldet, so z. B. das Taurus-Projekt der Londoner Börse. Begonnen wurde die Softwareentwicklung für die elektronische Wertpapierverwaltung bereits im Jahre 1980. 12 Jahre später, als das Projekt endlich abgebrochen wurde, hatte es schon 13-mal mehr gekostet, als ursprünglich geplant. Der Gesamtverlust für die City of London wird auf ca. 800 Millionen Pfund beziffert. Es sollte eine der ersten großen verteilten Anwendungen werden. Stattdessen wurde es eine der größten Pleiten aller Zeiten.

Glücklicherweise sind durch diese Fehlschläge keine Personen ums Leben gekommen, was von dem Londoner Ambulance-Service-Projekt nicht behauptet werden kann. Das System für die Einsatzleitung der Krankenwagen im Großraum London wurde gleich zweimal entwickelt. Die erste Entwicklung begann im Jahre 1987. Geplant wurde sie für drei Jahre mit einem Budget von 2,5 Millionen Pfund. 1990 wurde dieses Projekt nach zwei katastrophalen Fehlversuchen mit einem Verlust von 7,5 Millionen Pfund aufgegeben. Ein zweiter Versuch wurde 1991 mit einem anderen Softwarelieferanten gestartet. Diesmal wurden die Anforderungen reduziert und eine neue Entwicklungstechnologie eingesetzt. Ende 1992 wurde das neue System ausgeliefert. Nach mehreren Systemzusammenbrüchen, in deren Folge 46 Menschen starben, wurde auch diese Version aufgegeben.

In der kommerziellen Datenverarbeitung sind die Erfolgsaussichten, statistisch gesehen, geringer als in anderen Disziplinen einzuschätzen. Das liegt vor allem an der Schwierigkeit, die Anforderungen der künftigen betrieblichen Informationssysteme überhaupt zu erfassen. Eine Studie von Kweko Ewusi-Menach [Ewusi97] über Entwicklungsprojekte für Managementinformationssysteme (MIS) im US-Staat Kalifornien belegt, dass über 40 % aller MIS-Projekte vor dem Abschluss abgebrochen werden. Weitere 16 % dieser Projekte werden nur teilweise fertig. Dabei schneiden moderne Client/Server-Systeme besonders schlecht ab. Ihre Misserfolgsrate beträgt 62 %, d. h., von 100 angefangenen Cient/Server-Systemen werden nur 38 erfolgreich beendet.

Zusammenfassend zeichnen die Studien und eine Vielzahl weiterer hier nicht erwähnter Untersuchungen ein verheerendes Urteil über den Stand der Softwareentwicklung Ende der 90er Jahre. Offensichtlich wuchsen die Ansprüche an Software und damit deren Komplexität schneller als die verfügbaren und von den Entwicklungsteams beherrschten Methoden und Techniken.

## 1.2 Softwareentwicklung heute

Die durch den Milleniumswechsel (»Jahr-2000-Problem«) und die Euro-Umstellung sowie die durch oft zu kurzsichtig entworfene Software erforderlichen Wartungs- und Umstellungsarbeiten beanspruchten Ende der 90er Jahre und zu Beginn des neuen Jahrtausends einen großen Teil der personellen Ressourcen für sich. Der damit verbundene »Anwendungsstau« und der erhöhte Bedarf an internet- bzw. webgestützten Anwendungssystemen führten zu zwei gegenläufigen Trends in der Softwareentwicklung. Die geforderte immer kürzere Entwicklungszeit (... die Konkurrenz ist nur einen Mausklick entfernt) führte einerseits zu implementierungslastigen, auf Modellierung und zusätzliche Dokumentation weitestgehend verzichtenden Vorgehensweisen wie z.B. dem Extreme Programming [Beck99], während die Fortschritte bei vereinheitlichten Realisierungs- und Modellierungstechniken wie z.B. der Java Enterprise Edition (J2EE), Microsofts .NET-Rahmenwerk und der Unified Modeling Language (UML) insbesondere in objektorientierten Projekten die Erstellung von Anwendungssystemen unter Benutzung von Modellen in allen Phasen der Softwareentwicklung propagierten [BRJ99].

Wie hat sich in dieser Zeit die Erfolgswahrscheinlichkeit von Softwareprojekten entwickelt? Der im Jahr 2004 vorgelegte, auf 9.236 ausgewerteten Projekten basierende CHAOS-Forschungsbericht der Standish-Gruppe zeigte im Vergleich zu 1994 zwar einige positive Tendenzen, gab aber dennoch Anlass zur Nachdenklichkeit. Zwar waren 29% aller Projekte erfolgreich und nur 18% wurden vor der Fertigstellung aufgegeben, was einer ca. 90%igen Verbesserung gegenüber dem Stand von 1994 entspricht. Allerdings sind immer noch 53% der untersuchten Projekte kritisch, d.h. hatten Zeit oder Budget oder beides signifikant überschritten und/oder wurden mit erheblich reduziertem Funktionsumfang ausgeliefert.

Interessant ist ein näherer Blick auf die untersuchten Projekte. Während 36% der Projekte Neuentwicklungen mit traditionellen Sprachen und Methoden darstellten, handelte es sich bei 19% der Projekte um Neuentwicklungen mit objektorientierten Sprachen und Methoden und bei 16% um die Entwicklung und den Zukauf von mehreren Komponenten. 13% befassten sich mit dem Kauf und der Anpassung kompletter Anwendungssysteme, jeweils 6% mit dem Kauf einzelner Komponenten bzw. dem Kauf und der massiven Modifikation kompletter Anwendungssysteme und nur 4% aller berücksichtigten Projekte bezogen sich auf den Kauf kompletter Anwendungssysteme ohne weitere Anpassung. Im Mittelpunkt der betrachteten IT-Projekte stand somit nach wie vor die Entwicklung neuer Software.

In Deutschland zeichnet sich heute ein düsteres Bild ab. So kritisierte der Bundesrechungshof mehrere Großprojekte der Informationstechnik (IT) des Bundes und der Länder wegen schlampiger Planung, fehlender Steuerung und Erfolgskontrolle, inkom-

petenter Mitarbeiter, unflexibler Insellösungen und unausgereifter oder überfrachteter Konzepte:

- Das 1991 begonnene Softwareprojekt Fiskus, das den 650 Finanzämtern Deutschlands spätestens ab 2006 einheitliche Programme bringen sollte, wurde von den Finanzministern von Bund und Ländern in einer Sondersitzung im Juli 2004 für gescheitert erklärt. Die Programmierversuche haben den Steuerzahler nach internen Schätzungen bislang zwischen 250 und 900 Millionen Euro gekostet.
- Das Großprojekt »Toll Collect« sollte die Erhebung der LKW-Maut auf bundesdeutschen Autobahnen automatisieren. Nachdem im September 2002 entsprechende Aufträge an ein deutschfranzösisches Firmenkonsortium ergangen sind, wurde die herkömmliche Mauterhebung mit Vignetten unwiderruflich gekündigt und war zum geplanten Fertigstellungstermin des neuen Systems – Oktober 2003 – bereits abgewickelt. Wegen erheblicher Fehler konnte das neue System jedoch erst zum Januar 2005 in Betrieb genommen werden, was zu Einnahmeverlusten von 3,8 Milliarden Euro führte.
- Die Software zur Berechnung des zum Jahr 2005 neu geregelten Arbeitslosengeldes 2 (»Hartz IV-Software«) konnte nicht termingerecht eingesetzt werden. Auch die Ende Oktober 2004 ausgelieferte Software war noch unfertig und enthielt Fehler, die zu Falschzahlungen führen konnten. Außerdem ließen sich bestimmte Gruppen von Langzeitarbeitslosen gar nicht von der Software erfassen.

Wie können solche Misserfolge in Zukunft vermieden werden? Es lohnt sich, einen Blick auf die wesentlichen Erfolgsfaktoren von Softwareprojekten zu werfen [MBP+04]. Aus den CHAOS-Studien ergibt sich, dass an erster Stelle die Geschäftsführung einheitlich hinter den Entwicklungszielen und der Einführungsstrategie der Software stehen und den Projektleiter durch schnelle und konsequente Entscheidungen bei der Ressourcenzuteilung, beim Setzen von Prioritäten sowie bei der Lösung von Konflikten unterstützen muss. Es müssen klare Verantwortlichkeiten für die Projektkosten sowie die Festlegung, Kommunikation, Abstimmung, Umsetzung und Überprüfung der Projektziele inkl. des konkreten Nutzens herrschen.

Darüber hinaus ist es essenziell, dass zukünftige Benutzer des Systems die Entwicklung durch Vorgaben, Reviews, Erprobungen, Evaluation, Tests etc. begleiten. Auch der Projektleiter spielt eine Schlüsselrolle für den Erfolg des Softwareprojekts, da er die Schnittstelle zwischen Auftraggeber und Entwickler bildet und für den reibungslosen Ablauf des gesamten Projekts verantwortlich ist. Natürlich hat auch die Größe des Projekts bedeutenden Einfluss auf dessen Erfolg – je größer und komplexer das Projekt ist, desto schwerer ist es zu beherrschen und desto wahrscheinlicher ist sein Scheitern.

Da sich die Anforderungen an eine Software häufig ändern, sollte auf Seiten der Technik zumindest ihre Architektur ausgereift und stabil sein, damit die Softwareentwickler in diesem stabilen Rahmen auf Änderungen reagieren können. Zu Projektbeginn sollte ein minimaler Satz zentraler, möglichst stabiler Anforderungen festgelegt und in einem ersten Schritt realisiert werden, dessen Ergebnisse dann als Basis für weitere Entwicklungsschritte dienen. Last but not least sollte ein standardisierter Entwicklungs- und Wartungsprozess inklusive Qualitätssicherungsmaßnahmen etabliert sein, der auf spezifische Bedürfnisse der Projekte angepasst werden kann.

## 1.3 Was ist Software Engineering?

Die ersten beiden Abschnitte zeigten, dass eine Verbesserung der Situation bei der Softwareentwicklung nur durch den breiten Einsatz gesicherter Methoden des Software Engineering und durch kompetentes Personal erzielt werden kann. Was aber ist nun genau unter *Software Engineering* zu verstehen? Knapp gesprochen befasst sich Software Engineering mit der systematischen Entwicklung großer Softwaresysteme. Detaillierter formuliert beschäftigt sich Software Engineering mit Prinzipien, Methoden, Techniken und Werkzeugen zur Entwicklung großer Softwaresysteme mit dem Ziel, diese

- mit dem festgelegten Funktionsumfang,
- in hoher Qualität,
- kostengünstig innerhalb eines festen Budgetrahmens und
- zum geplanten Zeitpunkt

zu produzieren.

Diese Formulierung macht deutlich, dass Software Engineering nicht nur die Softwareentwicklung im engeren Sinn behandelt, sondern auch Fragen der Qualitätssicherung (es geht um die Entwicklung von Software hoher Qualität) und des Managements zum Gegenstand hat (es geht um die kostengünstige Entwicklung innerhalb einer fest vorgegebenen Zeit). Ein weiterer wichtiger Aspekt besteht darin, dass große Softwaresysteme von heterogen zusammengesetzten Teams entwickelt werden (die beteiligten Personen setzen sich u.a. aus Auftraggebern, Anwendungsexperten, Benutzern, Entwicklern und Managern zusammen), so dass auch interdisziplinäres Wissen und soziale Kompetenz der Beteiligten für den Projekterfolg eine zentrale Rolle spielen.

Software Engineering ist keine Modeerscheinung, sondern hat sich seit der Erkennung der »Softwarekrise« in den späten 1960er Jahren zu einer anerkannten Disziplin der Informatik entwickelt, deren Prinzipien, Methoden und Techniken in abertausenden von Konferenz- und Zeitschriftenartikeln und hunderten von Büchern dokumentiert sind. Gemäß dem oben Gesagten behandelt Software Engineering also die praktischen Probleme bei der Herstellung (der Software) von IT-Systemen, während sich die meisten anderen Disziplinen der Informatik mit Modellen und Theorien bezüglich bestimmter Teilaspekte der Analyse und Konstruktion solcher Systeme (Komplexität, Berechenbarkeit, Datenstrukturen, Algorithmen, ...) bzw. bestimmter Arten von Systemen (Betriebssysteme, Übersetzer, Datenbankmanagementsysteme, ...) beschäftigen.

Nach vielen Jahren voller Vorschläge, leidvoller Versuche und Irrtümer kann man mittlerweile von einer Konsolidierung des Software Engineering reden, die sich auch in offiziellen Dokumenten wie z.B. dem »Guide to the Software Engineering Body of Knowledge« (kurz: SWEBOK) des IEEE Computer Society Professional Practices Committee niederschlägt. Dort werden folgende Wissensgebiete des Software Engineering unterschieden [SWEBOK04]:

- Anforderungsermittlung (software requirements)
- Entwurf (software design)
- Realisierung (software construction)
- Testen (software testing)

- Wartung (software maintenance)
- Konfigurationsmanagement (software configuration management)
- Projektmanagement und Metriken (software engineering management)
- Vorgehensmodelle (software engineering process)
- Werkzeuge und Methoden (software engineering tools and methods)
- Softwarequalität (software quality)

Diese Aufzählung lässt erahnen, dass Software Engineering eine äußerst fassettenreiche Disziplin ist, die nicht nur vielfältige Bezüge zu anderen Fachgebieten der Informatik hat, sondern auch außerhalb der Informatik verwurzelt ist. So fließen z. B. Erkenntnisse der Ingenieur- und Wirtschaftswissenschaften bis hin zur Kommunikations- und Arbeitspsychologie in die Wissensgebiete des Software Engineering ein.

Während man lange Zeit dachte, dass Software Engineering unabhängig vom Einsatzbereich der zu erstellenden Software ist, lernte man diesbezüglich mit der Zeit wichtige Unterschiede kennen. So stehen z. B. bei eingebetteten Systemen Aspekte der Zuverlässigkeit und der Reaktionszeit im Vordergrund, während bei interaktiver Software deren Benutzbarkeit an die erste Stelle rückt. Durch neue Entwicklungen wie z. B. internetbasierte Systeme und Multimedia-Anwendungen mit mobilen Einsatzkontexten kristallisieren sich zudem ständig neue Gesichtspunkte heraus, deren Behandlung eigene Teildisziplinen erfordern. Heute existiert also nicht *das* Software Engineering, sondern es gibt verschiedene sinnvolle Ausprägungen für unterschiedliche anwendungs- und technologiebezogene Schwerpunktsetzungen [FloZül97].

*Softwareentwicklung* bezeichnet die Gesamtheit aller Aktivitäten, die zu einem lauffähigen Softwaresystem führen. Software Engineering befasst sich in diesem Rahmen damit, einzelne Aktivitäten zu identifizieren und zu beschreiben, ihre Ergebnisse festzulegen und in einem Vorgehensmodell anzuordnen. Dabei wird häufig nach *produktbezogenen* (eher technischen) und nach *prozessbezogenen* (eher nichttechnischen) *Aktivitäten* unterschieden. Produktbezogene Aktivitäten finden z. B. bei der Anforderungsermittlung und der Implementierung statt; ihre Ergebnisse gehen direkt in das Produkt ein – z. B. als definierende Dokumente oder kompilierbarer Code. Prozessbezogene Aktivitäten stellen den Organisations- und Managementrahmen für die produktbezogenen Aktivitäten in den Vordergrund und umfassen z. B. die Projektplanung, die Leitung und Kontrolle während der Projektdurchführung, die Produktverwaltung und das Qualitätsmanagement.

## 1.4 Eigenschaften von Software

Zu einem adäquaten Verständnis der Softwareentwicklung muss man sich zunächst die Eigenschaften von Software verdeutlichen. Zu den grundsätzlichen Merkmalen gehören:

- *Software ist verformbar*. Im Gegensatz zu vielen anderen Produkten wie z. B. Autos, Mobiltelefone oder Industrieanlagen sind nicht nur die Pläne bzw. Entwurfsdokumente, sondern auch das (End-)Produkt Software an sich leicht zu ändern. Daher erwartet man, dass Software auch einfach und schnell an veränderliche Bedürfnisse anpassbar ist. Allerdings können selbst kleinste Änderungen von Software fatale Folgen haben (s. u.).

- *Software ist digital.* Aufgrund ihrer wert- und zeitdiskreten Struktur greifen die Verfahren der traditionellen kontinuierlichen Ingenieurmathematik nicht. Die formale Behandlung beruht vielmehr auf der diskreten Mathematik und der Logik. Entsprechende formale mathematisch-logische Techniken sind allerdings nur für kleine Systeme bzw. Teile von Systemen praktikabel.
- *Software ist lediglich eine statische Beschreibung dynamischer Prozesse.* Ihr Nutzen erschließt sich erst durch ihre Ausführung auf einem Prozessor. Ihre für den Menschen lesbare Form (Modelle, Quelltexte) spiegelt diese Prozesse nur bedingt wider, was die Verständlichkeit erschwert. Zudem ist der Zusammenhang zwischen der Software und den aus ihrer Ausführung resultierenden Prozessen hochgradig nichtlinear und unstetig, so dass die Änderung auch nur eines Zeichens im Quelltext zu vollkommen anderen Prozessen führen kann.

Spricht man von Softwareentwicklung und Software Engineering, meint man normalerweise die Entwicklung großer Softwaresysteme. Groß ist hierbei z. B. ein Softwaresystem, an dem ein Team von mehr als fünf Personen mehr als ein Jahr entwickelt. Im Gegensatz dazu kann ein »kleines« Programm von bis zu fünf Personen in wenigen Monaten erstellt werden. Große Software weist folgende wesentliche Eigenschaften auf [FloZül97]:

- *Große Software ist komplex.* Sie verknüpft mehrere Realitätsbereiche mit ihrem Fachwissen und ihrer Fachsprache und unterstützt kompliziert strukturierte und miteinander interagierende Handlungen, die über einen langen Zeitraum oder räumlich weit verteilt ablaufen können. Daraus resultieren Systeme aus miteinander verknüpften Teilen, wobei sowohl die Verständlichkeit des Gesamtsystems als auch die der einzelnen Teile anzustreben ist. Ein zentrales Anliegen des Software Engineering besteht daher in der Beherrschung der Komplexität.
- *Große Software besteht aus umfangreichen semiotischen*[1] *Artefakten, z. B.* Spezifikationen, Programmen und unterschiedlichen Handbüchern. Die Artefakte liegen in einer grafischen Notation oder textuell vor. Auch sind sie nur teilweise formalisiert, so dass es keine gemeinsame verbindliche Semantik gibt. Ihre Konsistenz und Vollständigkeit kann daher insgesamt nicht überprüft werden.
- *Große Software hat eine lange Lebensdauer.* Langlebig bedeutet, dass die Software über mehrere Jahre (oft länger als 10 Jahre) eingesetzt wird, um den hohen Entwicklungs- und Einarbeitungsaufwand für die Benutzer zu amortisieren. Dabei muss sie kontinuierlich an sich ändernde Anforderungen (geänderte und erweiterte Funktionalität, andere Plattformen) angepasst werden. Dies führt zur Bildung von Versionen.
- *Große Software konstruiert Realität.* Sie ist in sozio-technische Systeme eingebettet, bildet also einerseits soziale Wirklichkeit in vielfältiger Form ab und ändert andererseits diese Wirklichkeit durch ihren Einsatz, indem sie neue Rahmenbedingungen für Arbeitsabläufe schafft. Große Software kann durch ihren Einsatz die an sie gestellten ursprünglichen Anforderungen in Frage stellen.

1. Semiotik ist die Lehre vom Austausch kodierter Informationen und den diesen zugrunde liegenden Zeichensystemen.

Die Entwicklung großer Software wirft somit grundlegend andere Probleme auf und braucht andere methodisch-technische Unterstützung als die kleiner Programme. Während bei vielen anderen Produkten die Anzahl der Entwickler wesentlich kleiner ist als die Anzahl der zur (Massen-)Produktion erforderlichen Mitarbeiter, kehrt sich dieses Verhältnis im Falle von großer Software um, da sich die »eigentliche« Produktion auf das Vervielfältigen von Datenträgern und Handbüchern beschränkt. Das Gleiche gilt für die Kosten, deren Großteil bei der Entwicklung anfällt, während die Produktionskosten eher gering sind. Darüber hinaus wird in der Softwareentwicklung für jedes Produkt in der Regel eine eigene Organisation mit Projektmitarbeitern, Infrastruktur etc. angelegt.

## 1.5 Anwendungssysteme und Geschäftsprozesse

*Anwendungssysteme* (oft kurz *Anwendungen* oder Applikationen genannt) sind Softwareprodukte, die in bestimmten Anwendungsbereichen eingesetzt werden und dort wiederholt auszuführende Arbeitsprozesse unterstützen (Finanzverwaltung, Auftragsbearbeitung, Lagerhaltung).

Ein Anwendungssystem kann eine *Individualsoftware* sein, die für einen speziellen Auftraggeber in einem bestimmten Anwendungsbereich erstellt wird. Hierunter fallen z. B. eine Software für die Raumplanung einer Hochschule oder die Website bzw. das Internetportal einer Firma. Ein Anwendungssystem kann aber auch eine *Standardsoftware* sein, die einen bestimmten Anwendungsbereich abdeckt und durch Konfiguration an die unterschiedlichen Bedürfnisse verschiedener Anwender angepasst werden kann. Hierzu gehören z. B. Finanzbuchhaltungs- und Materialwirtschaftssysteme, Branchenlösungen für Banken oder Handwerksbetriebe und Office-Programme für Textverarbeitung und Tabellenkalkulation.

Manchmal steuern Anwendungssysteme auch technische Prozesse (Flugüberwachung, Fertigungsstraßen). Mit *eingebetteten Systemen* (Motormanagement im Automobil, Steuerung des DVD-Players) wird teilweise schon die Grenze zu der nicht unter den Begriff Anwendungssystem fallenden *Systemsoftware* (Betriebssysteme, Übersetzer, Datenbanken) überschritten, die sich keinem bestimmten Anwendungsbereich zuordnen lässt.

Anwendungssysteme sind in Unternehmenskontexte eingebunden und unterstützen dort zusammenhängende Abläufe, die man als *Geschäftsprozesse* bezeichnet. Scheer definiert den Begriff Geschäftsprozess wie folgt [Scheer98]:

> Allgemein ist ein *Geschäftsprozess (business process)* eine zusammengehörige Abfolge von Unternehmensverrichtungen zum Zweck einer Leistungserstellung. Ausgang und Ergebnis des Geschäftsprozesses ist eine Leistung, die von einem internen oder externen »Kunden« angefordert und abgenommen wird.

*Geschäftsprozesse* sind komplexe fachliche Abläufe innerhalb eines oder zwischen mehreren Unternehmen, die zu einer Leistungserstellung führen. Sie beschreiben den kompletten Weg von der originären Leistungsanforderung bis zur Leistungsablieferung (*End-to-End-Charakteristik*). Oft erfolgt die Leistungsanforderung durch einen Kun-

den, an den dann auch die erbrachte Leistung abgeliefert wird. Geschäftsprozesse sind damit im Normalfall wesentlich umfangreicher und komplizierter als Anwendungsfälle (vgl. Abschnitt 12.2). Beispiele für Geschäftsprozesse sind Auftragsverfolgung, Bedarfsplanung, Lagerzugang oder Inventur.

Bevor man über die IT-Unterstützung bestehender Geschäftsprozesse nachdenkt, müssen diese in einem ersten Schritt selbst beschrieben, analysiert, spezifiziert und ggf. noch modifiziert bzw. optimiert werden, was oft unter den Begriff *Business Process Reengineering* gefasst wird (vgl. [HamCha94]). Erst auf der Grundlage wohlverstandener, begründeter und präzise spezifizierter Geschäftsprozesse ist es entscheidbar, inwieweit darin eingebettete menschliche Handlungen von Anwendungssystemen zu unterstützen sind. Manchmal werden auch völlig neuartige Geschäftsprozesse geschaffen, die ohne IT-Unterstützung nicht möglich wären. Hierbei sind die menschlichen Handlungen und die unterstützenden Anwendungssysteme gemeinsam zu gestalten.

Einen verbreiteten Ansatz zur Spezifikation von Geschäftsprozessen bilden *ereignisgesteuerte Prozessketten (EPK)*, die auch die Basis vieler Werkzeuge zur computergestützten Modellierung von Geschäftsprozessen darstellen (vgl. [Scheer02], [RicStu04]). Die Basiselemente einer EPK sind *(Geschäftsprozess-)Ereignisse* und *Funktionen* (kurz: GP-Ereignisse und GP-Funktionen).[1]

- Ein GP-Ereignis repräsentiert einen betriebswirtschaftlich relevanten Sachverhalt, der an einem bestimmten Zeitpunkt vorliegt und Einfluss auf einen oder mehrere Abläufe hat. Beispiele hierfür sind »Rohmaterial eingetroffen«, »Ware verpackt« oder »Informationen erfasst«.
- Eine GP-Funktion beschreibt eine Handlung, die in einem Zeitabschnitt ausgeführt wird, wie z. B. »Rohmaterial liefern«, »Ware verpacken« oder »Informationen erfassen«. Synonyme für GP-Funktionen sind z. B. Prozesse, Aktionen, Aktivitäten, Arbeitsschritte oder Tätigkeiten.

Jede GP-Funktion wird durch (mindestens) ein *Start-GP-Ereignis* ausgelöst, von Menschen und/oder Maschinen bearbeitet und durch (mindestens) ein *Ergebnis-GP-Ereignis* abgeschlossen. Eine Kette von sich abwechselnden Ereignissen und Funktionen entsteht dadurch, dass Ergebnis-Ereignisse von Funktionen zu Start-Ereignissen nachfolgender Funktionen werden. Die Reihenfolge von Modellelementen wird durch gerichtete Kanten von Vorgänger- zu Nachfolgerelementen verdeutlicht (vgl. Abb. 1–1).


***Abb. 1–1*** *Grafische Elemente ereignisgesteuerter Prozessketten*

Außer der reinen Aneinanderreihung von GP-Ereignissen und Funktionen können zur Erhöhung der Ausdrucksmächtigkeit *logische Verknüpfungsoperatoren* zwischen

1. Im Weiteren wird das Präfix GP oft weggelassen, wenn keine Verwechslungen mit (UML-)Ereignissen etc. zu befürchten sind.

Modellelementen verwendet werden. Die gerichteten Verbindungskanten und Verknüpfungsoperatoren spezifizieren die Ablaufregeln, nach denen die Geschäftsprozesse ausgeführt werden.

Die Spezifikation von Geschäftsprozessen umfasst im Allgemeinen mehr als nur die Verknüpfung von Ereignissen und Funktionen, sondern berücksichtigt auch deren Einbettung in die beteiligten Organisationseinheiten, das Konsumieren und Erzeugen von Leistungen sowie die Benutzung und Veränderung von Daten. Durch die Hinzunahme von Modellelementen für *(Aufbau-)Organisation*, *Leistung* und *Daten* entsteht die *erweiterte ereignisgesteuerte Prozesskette (eEPK)*. Für unsere Betrachtungen sind im Wesentlichen nur das zur Spezifikation der Aufbauorganisation gehörende Element *menschliche Ressource* (in ARIS: »Personentyp« oder »Stelle«) und dessen *Verbindung* (in ARIS: Beziehungstyp »führt aus«) zur zugehörigen Funktion relevant.

**Beispiel 1–1** Eine Kundenauftragsbearbeitung soll als Beispiel für einen Geschäftsprozess dienen [Scheer98]. Ein Kunde bestellt bei der Vertriebsabteilung eines Unternehmens einige Produkte, die gefertigt werden müssen. Die Bestellung wird anhand von Informationen über Kunden und Produkte auf ihre Ausführbarkeit geprüft. Bei Auftragsannahme werden die benötigten Materialien von entsprechenden Lieferanten beschafft. Nach Eintreffen der Materialien und Einplanung des Auftrags werden die Produkte anhand eines Arbeitsplans gefertigt und an den Kunden versandt. Zu jedem Produkt wird auch eine Produktinformation erstellt und versandt. Das Geschäftsprozessdiagramm in Abbildung 1–2 zeigt den Ablauf der Kundenbestellung.

Hierbei sind *Vertriebssachbearbeiter*, *Warendisponent* und *Lagerist* Personentypen bzw. Stellen, während *Kunde*, *Bestellung* und *Produkt* Daten darstellen. Das Zeichen ⊗ mit den Verbindungen zu den drei GP-Ereignissen *Neukunde*, *Kundendaten gültig* und *Kundendaten verändert* steht für den Exklusiv-Oder-Operator, d.h. genau eines der drei Ereignisse tritt ein. Das ⊗ mit nur einer Verbindung zum Ereignis *Kundendaten gültig* besagt, dass entweder genau eine der beiden vorangehenden GP-Funktionen *Kunden anlegen* und *Kunden ändern* ausgeführt und beendet sein muss, damit das Ereignis *Kundendaten gültig* eintritt, oder das Ereignis *Kundendaten gültig* bereits nach der Ausführung der Funktion *Kundendaten prüfen* vorgelegen hat und somit keine der beiden Funktionen auszuführen war.

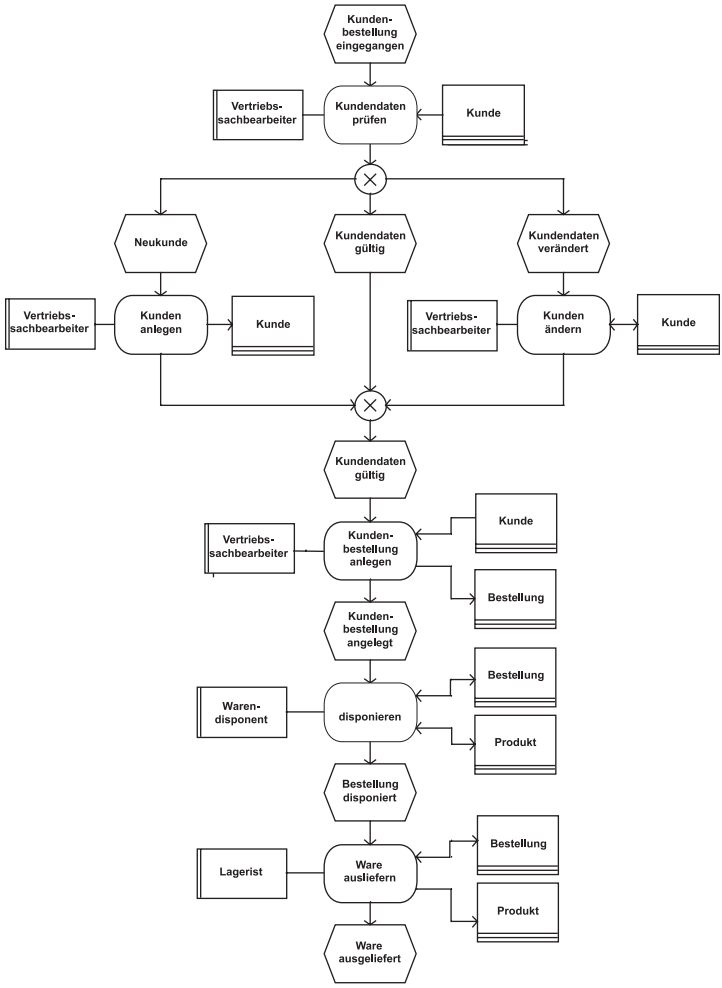


***Abb. 1–2*** *Erweiterte ereignisgesteuerte Prozesskette für den Geschäftsprozess »Ausführen einer Kundenbestellung«*

Geschäftsprozesse sind Gegenstand betriebswirtschaftlicher Betrachtungen, für die Ziele wie z.B. »Optimierung des zeitlichen Aufwands« definiert werden und die der Kosten- und Leistungsrechnung unterliegen. Wichtig ist das tiefe Verständnis der Anwendungssituation, was insbesondere eine große Anschaulichkeit der Modelle für Geschäftsprozesse bedingt.

Funktionen in Geschäftsprozessen können manuell, mit Softwareunterstützung oder vollautomatisch ausgeführt werden. Ziel der *Geschäftsprozessautomation* ist die Ablösung manuell ausgeführter Handlungen durch maschinell unterstützte oder vollautomatisch ausgeführte, sofern sich dadurch die Geschäftsprozesse schneller, effizienter und kostengünstiger gestalten lassen. Die Unterstützung oder Automatisierung (von

Teilen) eines Geschäftsprozesses durch ein Computersystem wird als *Workflow* bezeichnet [WFMC97], wobei es für betriebswirtschaftlich weitgehend standardisierte Workflows kommerzielle Systeme gibt, z. B. für Enterprise Resource Planning (ERP), Customer Relationship Management (CRM) und Supply Chain Management (SCM).

Ein System, mit dem man Geschäftsprozesse zunächst als *Workflow-Modell* spezifizieren kann und das die spezifizierten Geschäftsprozesse dann auch aktiv steuernd ausführt, nennt man *Workflow-Management-System* (WFMS) (vgl. z. B. [WFMC97], [RicStu04]). Workflow-Modelle besitzen somit eine größere Realisierungsnähe als Geschäftsprozesse, denn sie müssen alle Details der Geschäftsprozesse präzisieren und eine konsistente Gesamtbeschreibung liefern, die die folgenden Aspekte umfasst:

- Der *organisatorische Aspekt* zeigt, *wer* etwas ausführt bzw. ausführen kann (und darf).
- Der *funktionale Aspekt* spezifiziert, *was* ausgeführt wird.
- Der *operationale Aspekt* gibt an, *wie* etwas ausgeführt wird (z. B. computergestützt durch spezielle Anwendungssysteme oder manuell).
- Der *ablaufbezogene Aspekt* legt fest, *wann* oder besser *in welcher Reihenfolge* etwas ausgeführt wird; es wird also der »Kontrollfluss« spezifiziert.
- Der *informationsbezogene Aspekt* beschreibt das *Womit*, also die Daten.
- Der *kausale Aspekt* letztendlich gibt an, *warum* bzw. aufgrund welcher Intra-Workflow-Abhängigkeiten etwas ausgeführt wird.

Ein WFMS umfasst eine Komponente zur Modellierung der Geschäftsprozesse, eine Ablaufsteuerung zu ihrer Ausführung, eine Komponente zur Aktivierung und Informationsversorgung der beteiligten Anwendungssysteme sowie eine Auswertungskomponente zur Erhebung von Kennzahlen zur Prozessbewertung und Optimierung. Bei der Ausführung eines Geschäftsprozesses verbindet das WFMS dessen Funktionen gemäß der vorgegebenen Ablaufregeln. Nicht jeder Übergang von einer Funktion zur jeweiligen Nachfolgerfunktion kann vom WFMS automatisch vollzogen werden, z. B. wenn das Ergebnis-Ereignis der Vorgängerfunktion nicht maschinell ausgewertet werden kann. In solchen Fällen muss der Benutzer eingreifen und abhängig vom eingetretenen Ergebnis-Ereignis die nächste Funktion bestimmen.

Werden die Geschäftsprozesse eines Unternehmens von mehreren Anwendungssystemen unterstützt, kann ein WFMS eingesetzt werden, welches die koordinierte Ausführung der Anwendungssysteme gemäß des Workflow-Modells sowie ggf. eine anwendungsneutrale Datenhaltung ermöglicht. Ansätze zur Koordinierung bzw. Integration solcher Anwendungssysteme firmieren auch unter dem Begriff *Enterprise Application Integration*. Diesen Zusammenhang zeigt Abbildung 1–3.

Bezogen auf das obige Geschäftsprozessbeispiel der Kundenauftragsbearbeitung könnten die verschiedenen Anwendungssysteme zur Bearbeitung eines Kundenauftrags z. B. dedizierte Systeme für den Vertrieb, die Beschaffung, die Fertigung und den Versand darstellen.

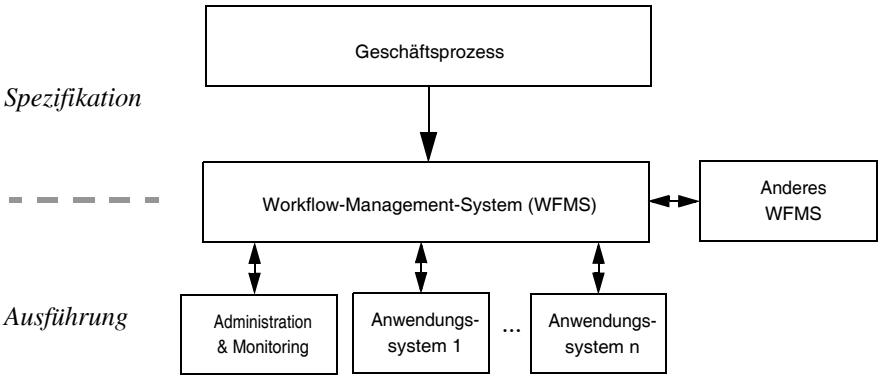


***Abb. 1–3*** *Enterprise Application Integration mit WFMS und Anwendungssystemen*