7. Spezifikation von Anforderungen

7.1 Grundlagen und Motivation

7.1.1 Definitionen und grundlegende Begriffe

DEFINITION 7.1. *Anforderung (requirement)*. (1) Eine Bedingung oder Fähigkeit, die von einer Person zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt wird. (2) Eine Bedingung oder Fähigkeit, die eine Software erfüllen oder besitzen muss, um einen Vertrag, eine Norm oder ein anderes, formell bestimmtes Dokument zu erfüllen. (IEEE 610.12-1990)

DEFINITION 7.2. *Anforderungsspezifikation*. Die Zusammenstellung aller Anforderungen an eine Software. Synonyme: Anforderungsdokument, Software Requirements Specification.

Im Alltag ist die Sprechweise nicht immer ganz eindeutig: mit „die Spezifikation“ ist häufig je nach Kontext das resultierende *Dokument* oder der *Spezifikationsprozess* (das heißt der Prozess des Erfassens, Beschreibens und Prüfens von Anforderungen) gemeint.

Werden die Aufgaben im Bereich der Spezifikation und des Spezifizierens systematisch und gezielt angegangen, spricht man auch von *Requirements Engineering* (Anforderungstechnik). In Definition 7.3 geben wir drei mögliche Definitionen, eine mehr technische, welche sich an die Definition von Software Engineering in IEEE 610.12-1990 anlehnt, eine mehr menschenorientierte, ähnlich derjenigen von Gause und Weinberg (1989), und schließlich eine risikoorientierte, welche auf Kosten- und Nutzenüberlegungen basiert.

DEFINITION 7.3. *Requirements Engineering (Anforderungstechnik)*. 1. Das *systematische, disziplinierte* und *quantitativ erfassbare* Vorgehen beim Spezifizieren, d.h. Erfassen, Beschreiben und Prüfen von Anforderungen an Software. 2. *Verstehen* und *Beschreiben*, was die Kunden *wünschen* oder *brauchen*. 3. Spezifikation und Verwaltung von Anforderungen mit dem Ziel, *das Risiko zu minimieren*, dass Software entwickelt wird, welche den Kunden nicht nützt oder gefällt.

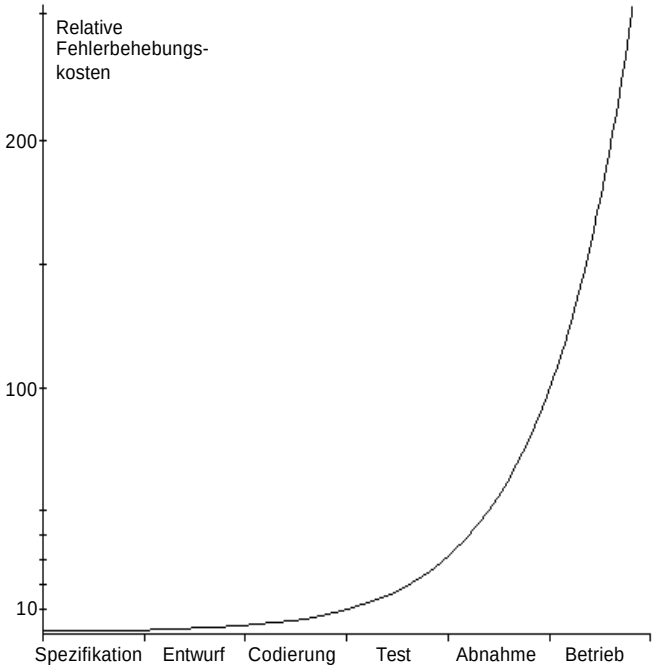
Im deutschen Sprachraum ist im Zusammenhang mit Anforderungen auch der Name *Pflichtenheft* gebräuchlich. Mit diesem Namen werden jedoch häufig verschiedene Begriffe verbunden. Manche Leute verstehen Pflichtenheft als ein Synonym zu Anforderungsspezifikation, andere verlangen eine grobe Beschreibung der gewählten Lösung als Bestandteil, wiederum andere sehen auch Elemente der Projektabwicklung (Kosten, Termine) als Bestandteil des Pflichtenhefts. Bei der Verwendung des Namens „Pflichtenheft“ ist daher Vorsicht geboten und das jeweils damit Gemeinte im Einzelfall zu klären.

7.1.2 Warum und wie viel Anforderungsspezifikation?

Die Erstellung einer Anforderungsspezifikation kostet Geld, ohne dass diesem Aufwand ein unmittelbar sichtbarer Ertrag in Form von Programmen gegenübersteht. Das Spezifizieren von Anforderungen ist also nur dann wirtschaftlich, wenn dem dafür zu treibenden Aufwand entsprechende Einsparungen gegenüberstehen.

Requirements Engineering, das systematische Spezifizieren von Anforderungen, hat daher das klare Ziel, *Kosten zu senken*.

Dass dieses Ziel realistisch ist, zeigt folgende Überlegung: *Fehlerkosten*, d.h. die Kosten für die Lokalisierung und Behebung von Fehlern, machen einen wesentlichen Teil der Gesamtkosten einer Systementwicklung aus. Anforderungsfehler sind dabei die teuersten Fehler, weil sie beim Fehlen einer Anforderungsspezifikation typisch erst bei der Abnahme oder im Betrieb gefunden werden und die Fehlerkosten exponentiell mit der Verweildauer der Fehler im System wachsen (Bild 7.1). Wenn man Requirements Engineering vernünftig betreibt, sind die Einsparungen bei den Fehlerkosten höher als der dafür notwendige Aufwand. Das heißt, das Spezifizieren von Anforderungen ist *wirtschaftlich* (Bild 7.2).



nach Boehm (1981)

BILD 7.1. Kosten für die Behebung von Fehlern abhängig von ihrer Verweildauer in der Software

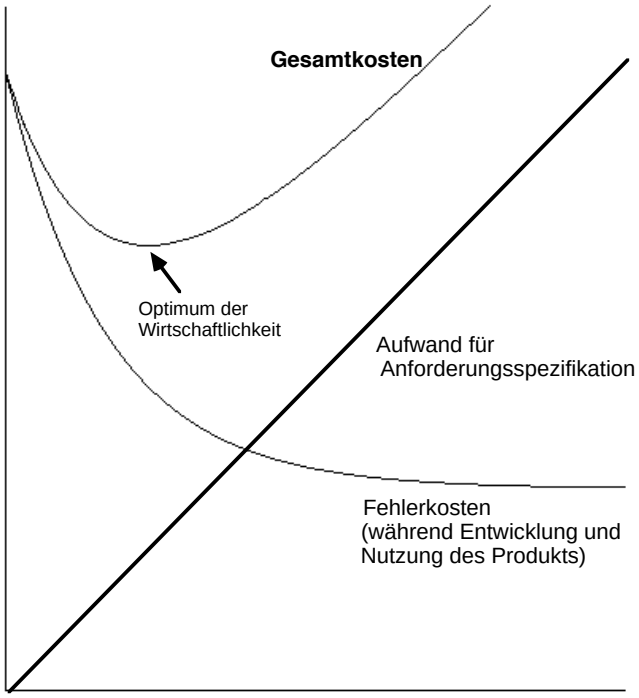


BILD 7.2. Wirtschaftlichkeit der Anforderungsspezifikation

Weitere wirtschaftliche Vorteile ergeben sich dadurch, dass mit Hilfe einer sorgfältigen Anforderungsspezifikation die Kundenzufriedenheit gesteigert werden kann, indem die Erwartungen der Kunden besser erfüllt und zuverlässigere Kosten- und Terminprognosen abgegeben werden können.

Die Tatsache, dass die Erstellung einer Anforderungsspezifikation grundsätzlich wirtschaftlich ist, sagt noch nichts darüber, wie viel Aufwand im Einzelfall in die Spezifikation von Anforderungen gesteckt werden soll. Hier helfen Risikoüberlegungen weiter (vgl. die dritte Definition in Definition 7.3). Wenn durch die Spezifikation von Anforderungen das Risiko minimiert werden soll, dass Software entwickelt wird, welche den Kunden nicht nützt oder gefällt, so bedeutet dies, dass in die Anforderungsspezifikation gerade so viel Aufwand investiert werden sollte, dass dieses Risiko eine vernünftige Größe annimmt.

REGEL 7.1. Der *Aufwand* für das Requirements Engineering soll *umgekehrt proportional* zum *Risiko* sein, das man bereit ist, einzugehen.

7.1.3 Merkmale einer guten Spezifikation

Um die Fehlerkosten zu senken, müssen wir folglich erreichen, dass (a) wenig Anforderungsfehler gemacht werden und (b) möglichst viele der dennoch gemachten Fehler möglichst früh gefunden werden. Dafür brauchen wir sowohl eine *gute Anforderungsspezifikation* als auch einen *guten Spezifikationsprozess*.

Eine gute Spezifikation zeichnet sich durch folgende Eigenschaften aus:
- Adäquatheit – das beschreiben, was der Kunde will bzw. braucht
- Vollständigkeit – alles beschreiben, was der Kunde will bzw. braucht
- Widerspruchsfreiheit – sonst ist die Spezifikation nicht realisierbar
- Verständlichkeit – für den Kunden und für die Informatiker
- Eindeutigkeit – damit Fehler durch Fehlinterpretationen vermieden werden
- Prüfbarkeit – damit feststellbar ist, ob das realisierte System die Anforderungen erfüllt
- Risikoorientierung – damit Aufwand und Risiko im richtigen Verhältnis zueinander stehen.

Ein guter Spezifikationsprozess ist charakterisiert durch
- Kundenorientierung
- Methodisches und zielgerichtetes Vorgehen
- Verwendung geeigneter Mittel
- Integration von Erstellung und Prüfung von Anforderungen.

7.1.4 Einbettung und Abgrenzung

Der weitaus größte Anteil von Software-Entwicklung entfällt auf Systeme, welche Anwendungsprobleme betrieblicher oder technischer Natur lösen bzw. Menschen oder Maschinen bei der Lösung solcher Probleme unterstützen (Software vom E-Typ, vgl. Kapitel 3.1.4). Bei der Spezifikation der Anforderungen an solche Systeme ist es von erheblicher Bedeutung, sich darüber klar zu werden, wie ein solches System in sein Umfeld eingebettet sein soll und wo die Systemgrenzen liegen. Bild 7.3 veranschaulicht die Situation. Ein Software-System kann man sich als eine spezialisierte Problemlösungsmaschine vorstellen, welche über Schnittstellen mit ihrer Umwelt interagiert. Ein solches System ist typisch in einem Anwendungsbereich situiert. Innerhalb dieses Anwendungsbereichs hat es einen so genannten *Kontext*; das sind diejenigen Elemente des Anwendungsbereichs, mit denen das System kommuniziert, die aber außerhalb des Systems liegen. Die Aufgabe, Anforderungen an ein solches System zu spezifizieren, bedeutet daher letztlich, den Kontext zu identifizieren, Anstöße aus dem Kontext und die geforderten

Reaktionen des Systems darauf zu spezifizieren und zu untersuchen, unter welchen Restriktionen diese Reaktionen zu erbringen sind. Dabei ist zu berücksichtigen, dass das Software-System (wenn es nach seiner Fertigstellung eingesetzt wird) die Realität verändert, d.h. dass gewisse Elemente des Anwendungsbereichs oder Beziehungen zwischen ihnen nachher nicht mehr oder nur noch in veränderter Form existieren.

Bei der Identifikation des Kontextes kommt es entscheidend auf die Betrachtungsebene an, und in vielen Projekten gibt es mehr als eine solche Ebene. Typisch gibt es mindestens eine Geschäfts-, eine System- und eine Softwareebene.

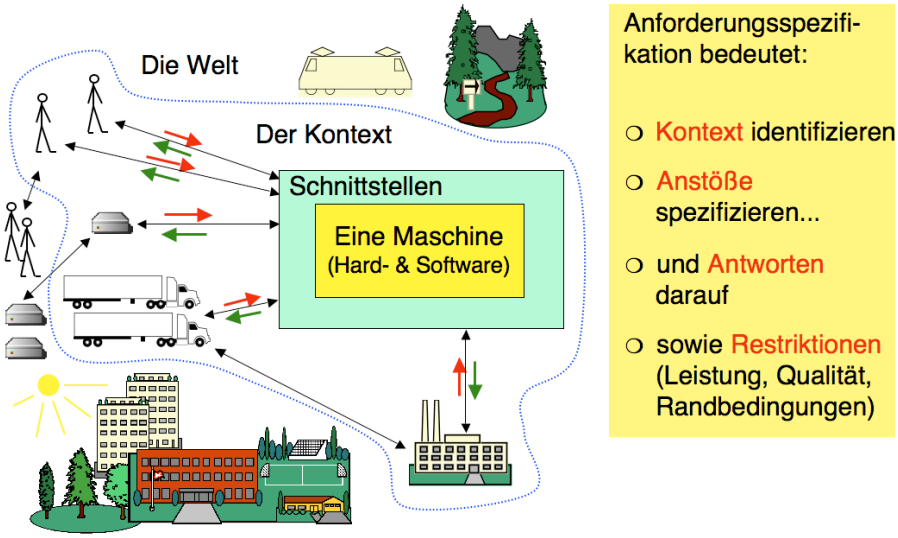


BILD 7.3. System und Kontext

7.2 Der Spezifikationsprozess

Im Requirements Engineering gibt es zwei Hauptprozesse: das Spezifizieren von Anforderungen und das Verwalten von Anforderungen.

Der Prozess des Spezifizierens von Anforderungen umfasst im Wesentlichen folgende Aufgaben: die Gewinnung (elicitation), die Analyse (analysis), die Dokumentation (documentation) und die Prüfung (validation) der Anforderungen. Allerdings gibt es keinen idealen Einheitsprozess, sondern der Spezifikationsprozess muss auf die jeweilige Projektsituation zugeschnitten werden. Wichtige Kriterien dabei sind:

- Wird das Projekt als Ganzes nach einem linearen oder einem inkrementellen Prozess abgewickelt?
- Muss die Spezifikation „wasserdicht“ sein, das heißt so gestaltet, dass sie als Vertrag für eine Realisierung der Software durch an der Spezifikation nicht beteiligte Dritte verwendet werden kann?
- Sind die Kunden und zukünftigen Benutzer bekannt und können sie in die Erstellung der Spezifikation einbezogen werden?
- Wird das zu spezifizierende System im Kundenauftrag oder für den Markt entwickelt?
- Soll für die Lösung Standardsoftware zum Einsatz kommen? (In diesem Fall müssen sich die Anforderungen zu erheblichen Teilen nach den Fähigkeiten der Standardsoftware richten.)

Ein guter Spezifikationsprozess kann nicht einfach sequenziell in die Schritte Gewinnung, Analyse, Dokumentation und Prüfung unterteilt werden. Vielmehr muss der Prozess iterativ in enger und ständiger Interaktion zwischen Vertretern des Kunden und den die Spezifikation erstellenden Informatikern/Analytikern ablaufen (Bild 7.4).

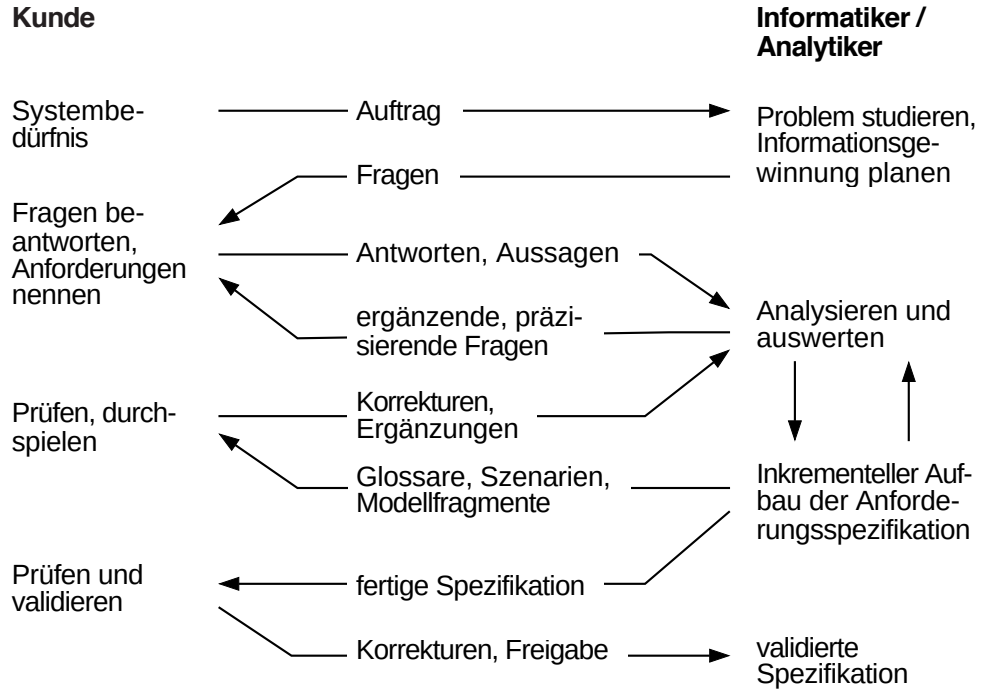


BILD 7.4. Möglicher Ablauf des Spezifikationsprozesses und Interaktionen der Beteiligten

7.3 Dokumentation von Anforderungen

7.3.1 Klassifikation von Anforderungen

Es gibt verschiedene Arten von Anforderungen (Bild 7.5). Zunächst einmal wird zwischen *Projekt-* und *Produktanforderungen* unterschieden. In diesem Kapitel betrachten wir nur letztere. Die Produktanforderungen wiederum gliedern sich in funktionale Anforderungen und Attribute. Unter *funktionalen Anforderungen* verstehen wir alle Anforderungen, die sich auf die Funktionalität eines Systems beziehen, das heißt, welche Ergebnisse aufgrund welcher Daten zu berechnen und/oder zu liefern sind. Die *Attribute* spezifizieren die Art und Weise, in der diese Funktionalität zu erbringen ist. *Leistungsanforderungen* sind Forderungen bezüglich Zeiten, Mengen, Geschwindigkeiten, Raten, etc. *Besondere Qualitätsanforderungen* sind Forderungen beispielsweise an Zuverlässigkeit oder Benutzerfreundlichkeit. *Randbedingungen* schließlich sind alle Forderungen, welche die Menge der möglichen Lösungen zusätzlich beschränken, z.B. Gesetze und Normen. Leistungsanforderungen, besondere Qualitätsanforderungen und Randbedingungen werden auch *nicht-funktionale* Anforderungen genannt.

Zusätzlich müssen die Anforderungen oft nach ihrer Wichtigkeit *priorisiert* werden, z.B. in

- *Muss-Anforderungen* – sind unverzichtbar und müssen in jedem Fall erfüllt werden
- *Soll-Anforderungen* – sollten erfüllt werden, sind aber bei zu hohen Kosten verzichtbar
- *Wunsch-Anforderungen* – werden nur erfüllt, wenn dies mit vertretbaren Kosten möglich ist.

Eine solche Priorisierung ist vor allem in zwei Situationen nötig:

- wenn die Entwicklungskosten und/oder Termine harte Randbedingungen darstellen. Dies ist unter anderem dann der Fall, wenn Software für den Markt entwickelt wird.
- wenn die Anforderungsspezifikation als Grundlage für die Beschaffung eines Systems dient, weil dann die Lösung nicht nach Maß auf die Bedürfnisse zugeschnitten werden kann.

Priorisierung ist ferner nützlich, wenn es bei inkrementeller Entwicklung (vgl. Kapitel 3.2.3) darum geht, Inhalt und Umfang der einzelnen Inkremente bzw. Teillieferungen festzulegen sowie bei der Releaseplanung im Rahmen der Weiterentwicklung bestehender Systeme (vgl. Kapitel 3.5).

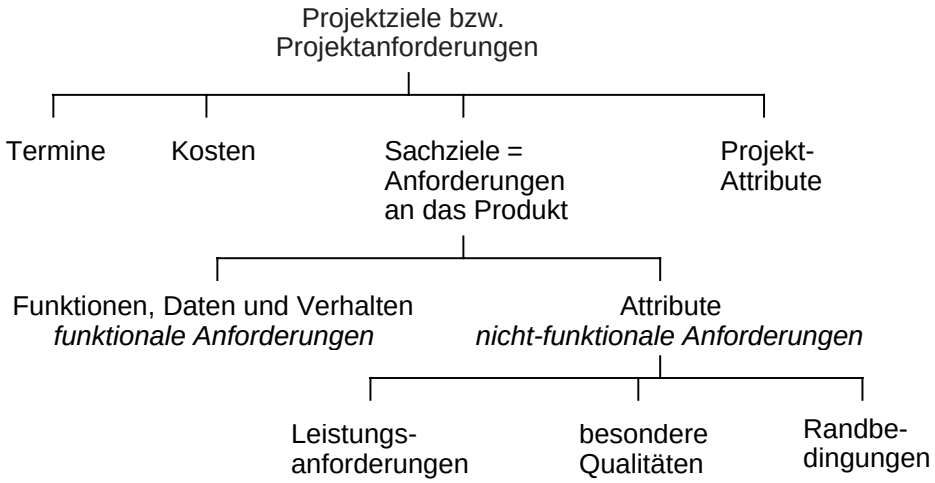


BILD 7.5. Gliederungsbaum der Anforderungen

7.3.2 Inhalt und Aufbau einer Anforderungsspezifikation

Eine Anforderungsspezifikation muss *inhaltlich* die folgenden Aspekte abdecken:

- Funktionaler Aspekt
  - Daten: Struktur, Verwendung, Erzeugung, Speicherung, Übertragung, Veränderung
  - Funktionen: Ausgabe, Verarbeitung, Eingabe von Daten
  - Verhalten: Sichtbares dynamisches Systemverhalten, Zusammenspiel der Funktionen
  - Fehler: Normalfall und Fehlerfälle
- Leistungsaspekt
  - Datenmengen (durchschnittlich/im Extremfall)
  - Verarbeitungs- /Reaktionsgeschwindigkeit (durchschnittlich/im Extremfall)
  - Verarbeitungszeiten und -intervalle
  - wo immer möglich: *messbare* Angaben!
- Qualitätsaspekt
  - geforderte Qualitäten (z.B. Benutzerfreundlichkeit, Zuverlässigkeit)
- Randbedingungsaspekt
  - einzuhaltende/zu verwendende Schnittstellen
  - Normen und Gesetze
  - Datenschutz, Datensicherung
  - Explizite Vorgaben des Auftraggebers.

Die *Gliederung* des Dokuments und die Art der Darstellung in den einzelnen Kapiteln hängen stark von den verwendeten Methoden und Sprachen ab. Teilweise sind sie auch durch Richtlinien des Kunden oder des entwickelnden Unternehmens oder durch die Anwendung internationaler Normen (z.B. IEEE 830-1993) bestimmt.

Bei der Wahl der Gliederung ist dem Qualitätsmerkmal der Verständlichkeit besonderes Augenmerk zu widmen. Schlechte oder gar nicht vorhandene Gliederungen behindern die Verständlichkeit unter Umständen massiv.

Die Qualität der Darstellung lässt sich ferner durch die Einhaltung einiger Regeln verbessern:

- Einzelanforderungen in kleinen Einheiten fassen: pro Einzelanforderung nur eine Kernaussage formulieren,

- Zu jedem geforderten Resultat die Funktion und die Eingabedaten, welche das Resultat erzeugen, spezifizieren,
- Mögliche Ausnahmesituationen spezifizieren: zu fast jedem Normalfall gibt es mögliche Ausnahmefälle,
- Implizite Annahmen aufdecken und explizit formulieren,
- Leistungs- und Qualitätsanforderungen quantitativ spezifizieren; nur so sind sie überpüfbar,
- All-Quantifizierungen kritisch hinterfragen: Aussagen der Art „Für alle Eingaben...“, „Jeder Alarm...“, usw. gelten so gut wie nie ohne Ausnahmen,
- Redundanz möglichst vermeiden und sie nur dort verwenden, wo sie für ein besseres Verständnis notwendig ist,
- Präzise spezifizieren, wobei sich der Grad der Präzision danach richtet, wie hoch das Risiko bei Missverständnissen und Fehlinterpretationen der jeweiligen Anforderungen ist.

7.4 Gewinnung und Analyse von Anforderungen

7.4.1 Grundlagen

Für die Gewinnung von Anforderungen (requirements elicitation) gibt es im Wesentlichen drei mögliche Quellen: die Befragung von Personen, die Beobachtung von Prozessabläufen und das Studium einschlägiger Unterlagen.

Um Personen befragen zu können, muss als erstes analysiert werden, wer in welcher Rolle Anforderungen an das zu spezifizierende System stellen kann. Solche Personen bzw. Rollen nennt man Beteiligte (stakeholders). Dazu gehören beispielsweise Endbenutzer, Auftraggeber, Betreiber und Projektleiter. Je nach Projekt kann es aber eine Vielzahl weiterer Beteiligter geben, beispielsweise Regulierungsbehörden.

Bei der Beobachtung von Prozessabläufen geht es darum, aus den IST-Abläufen einerseits ein Verständnis des Anwendungsbereichs zu gewinnen und andererseits Stärken und Schwächen des IST-Zustands zu erheben und daraus Anforderungen an das zu spezifizierende System zu gewinnen. Die Beobachtung von Prozessabläufen gibt ferner Aufschluss über Daten, die in diesen Prozessen benötigt werden. Daraus ergeben sich Anforderungen, welche Daten das zu spezifizierende System kennen muss.

Schließlich lassen sich aus dem Studium von Unterlagen, beispielsweise Ausschreibungstexten, Visionsdokumenten o.ä. häufig übergeordnete Ziele, Rahmenanforderungen und Randbedingungen gewinnen.

7.4.2 Techniken der Informationsbeschaffung

Um die gewünschten Informationen zu erhalten, können verschiedene Techniken eingesetzt werden.

- In *Interviews* werden die Beteiligten einzeln oder in kleinen Gruppen befragt.
- Mit *Fragebogen* können Begriffswelt und Bedürfnisse einer größeren Gruppe von Beteiligten erfasst werden.
- Durch *Beobachtung* von Beteiligten *bei der Arbeit* (und ggf. Stellen gezielter Fragen) können Prozessabläufe verstanden und Bedürfnisse der betreffenden Beteiligten erkannt werden.
- In *gemeinsamen Arbeitstagungen* (manchmal auch Joint Application Development-Sitzungen genannt) kann eine Gruppe ausgewählter Beteiligter und Analytiker gemeinsam die Anforderungen an ein geplantes System erarbeiten.

- Mit dem Bau und der Erprobung von *Prototypen* können mögliche Lösungen in der geplanten Einsatzumgebung auf ihre Tauglichkeit untersucht und daraus Anforderungen gewonnen werden.

7.4.3 Methodische Ansätze für die Gewinnung und Analyse von Anforderungen

Die folgenden Ansätze können erfolgreich zur Gewinnung und Analyse von Anforderungen eingesetzt werden. Je nach Situation wird eine einzelne Technik oder eine Kombination von Techniken gewählt. Es lassen sich *strukturorientierte* und *prozessorientierte* Ansätze unterscheiden (Bild 7.6).

- *Begriffe klären* und ein *Glossar* mit Definitionen der wichtigen Begriffe des Anwendungsbereichs erstellen. Damit wird eine begriffliche Grundlage geschaffen, die sicherstellt, dass alle das Gleiche meinen, wenn sie vom Gleichen reden. Ein solches Glossar ist insbesondere dann wichtig, wenn schon die Kundenvertreter untereinander keine klare und einheitliche Begriffswelt haben.
- *Geschäfts- und Datenobjekte analysieren*. Feststellen, welche Gegenstände der Anwendung für das zu spezifizierende System relevant sind. (Relevant sind diejenigen Gegenstände, über die das System Information speichern muss, damit es seine Aufgaben erfüllen kann.) Herausfinden, welche Eigenschaften der relevanten Gegenstände und welche Beziehungen der Gegenstände untereinander dem zu spezifizierenden System bekannt sein müssen.
- *Problem in Teilprobleme* zerlegen. Häufig lässt sich ein Problem in natürlicher Weise in eine Menge jeweils in sich weitgehend geschlossener Teilprobleme zerlegen. Da die Teilprobleme kleiner und weniger komplex sind als das Gesamtproblem, sind sie leichter zu analysieren.
- *Soll-Prozessabläufe untersuchen*. Feststellen, welche Ereignisse im Ablauf eines Prozesses auf das zu spezifizierende System einwirken und erfragen, wie das System auf welches dieser Ereignisse reagieren soll.
- *Dynamisches Systemverhalten untersuchen*. Viele Systeme kennen mehrere Zustände, in denen sie sich unterschiedlich verhalten. Es ist zu untersuchen, welche Zustände ein zu spezifizierendes System hat, welche Ereignisse welche Zustandsübergänge auslösen und welche Aktionen als Reaktion auf Zustandsübergänge auszuführen sind.
- *Anwendungsszenarien bilden und durchspielen*. Alle Interaktionen der Umgebung (seien dies Menschen oder Sensoren und Stellglieder) mit dem zu spezifizierenden System werden in Form von Szenarien aufgeschrieben und durchgespielt. Szenarien eignen sich besonders gut zur interaktiven Gewinnung und Diskussion von Anforderungen mit den Beteiligten.

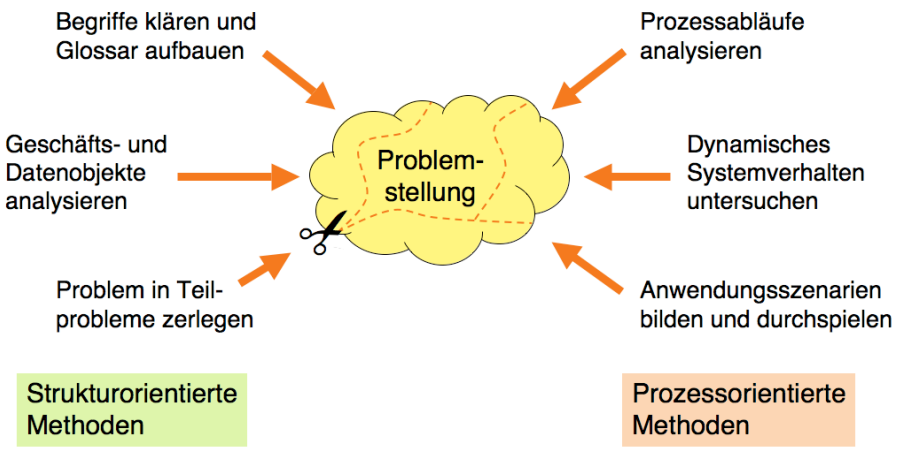


BILD 7.6. Ansätze zur Anforderungsgewinnung

Die Ergebnisse der Analyse werden mittels geeigneter Methoden und Notationen dargestellt und bilden dann die Anforderungsspezifikation. Es wäre jedoch falsch, in der Anforderungsgewinnung zuerst große Materialsammlungen anzulegen, diese dann zu analysieren und erst zum Schluss die gefunden Anforderungen zu dokumentieren. Stattdessen müssen Gewinnung, Analyse und Dokumentation *miteinander verzahnt* ablaufen (Bild 7.4). Die Anforderungsspezifikation wird fortlaufend inkrementell aufgebaut, wobei Lücken, Fehler und Schwachstellen erkannt und fortlaufend behoben werden können. Hierzu ist es wichtig, die niedergeschriebenen Spezifikationsfragmente fortlaufend an die Beteiligten zurückzuspielen und Rückmeldung von ihnen einzuholen (so genannter Autor-Kritiker-Zyklus). Ferner empfiehlt es sich, bei der Analyse möglichst mit Bekanntem und Gesichertem zu beginnen und sich von dort zum Unbekannten und Offenen vorzuarbeiten.

Bei der Gewinnung von Anforderungen treten typisch die folgenden *Schwierigkeiten* auf:

- Unterschiedliche Beteiligte haben unterschiedliche Vorstellungen über das zu spezifizierende System. Häufig sind schon die Auffassungen und die Begriffsbildung im Anwendungsbereich nicht einheitlich. Requirements Engineering beinhaltet daher auch *Konsensbildung* zwischen divergierenden Vorstellungen der beteiligten Personen.
- Die Beteiligten haben zwar eine Vorstellung, was sie wollen, aber sie können ihre Vorstellung nicht formulieren. Manchmal „lösen“ sie dieses Problem, indem sie anstelle ihrer eigentlichen Anforderungen bestehende Lösungen mit vergleichbaren Eigenschaften beschreiben.
- Die Beteiligten wissen nicht oder nur sehr vage, was sie eigentlich wollen. In solchen Situationen kann sich der Einsatz von Prototypen als sehr nützlich erweisen.

Requirements Engineering bedeutet daher immer auch, dass die Aufgabe bzw. das Problem genauer geklärt werden, Risiken zu analysieren sind, Konflikte erkannt werden müssen und ein Konsens zwischen verschiedenen Beteiligten gesucht wird. Dort, wo die Beteiligten nicht recht wissen, was sie wollen oder wo Produkte für den Markt zu entwickelt sind, muss ferner versucht werden, kreative Lösungen vorzuschlagen bzw. die Kreativität der Beteiligten anzuregen.

7.4.4 Die Rolle der IST-Analyse

In vielen Lehrbüchern, vor allem solchen älteren Datums (klassisch zum Beispiel in McMenamin und Palmer 1984), wird verlangt, dass der Spezifikationsprozess mit einer Analyse des IST-Zustands beginnt. Dabei wird folgendes Vorgehen empfohlen:

1 Analyse des IST-Systems; erstellen eines Modells der gegenwärtigen *Implementierung des IST-Systems*.
2 Analysieren der dieser Implementierung zugrunde liegenden Konzepte; erstellen des sogenannten *essentiellen Modells des IST-Systems*. Dabei wird von allen implementierungsspezifischen Eigenschaften des IST-Systems abstrahiert.
3 Ableiten der Anforderungen an das neue System; erstellen des *essentiellen Modells des SOLL-Systems*. Dieses Modell beschreibt die Anforderungen und ist im Idealfall frei von Entwurfs- und Implementierungsüberlegungen.
4 Entwurfs des SOLL-Systems; Erstellen des *Implementierungsmodells des SOLL-Systems*.

Der dritte Schritt in diesem Vorgehen ist der eigentliche Prozess der Anforderungsspezifikation. Dieses Vorgehen hält jedoch in vielen Fällen einer Wirtschaftlichkeitsprüfung nicht stand. Die Beschäftigung mit dem IST-Zustand ist nur dann gerechtfertigt, wenn

- die Informatiker/Analytiker die IST-Aufgaben und die IST-Arbeitsweise der Benutzer erst kennen lernen und verstehen müssen, damit sinnvolle Gespräche über das SOLL geführt werden können
- die Stärken und Schwächen des IST-Systems nicht bekannt sind

- Teile des IST-Systems übernommen werden sollen und festgestellt werden muss, welche Teile das sind
- das IST-System eins zu eins übernommen werden soll, beispielsweise, wenn ein System auf eine andere Rechnerplattform übertragen werden muss.

Hierzu ist es in der Regel nicht erforderlich, je ein vollständiges Implementierungsmodell und essentielles Modell zu erstellen. Jede über das Notwendige hinausgehende Beschäftigung mit dem IST-System verursacht Kosten, denen kein Nutzen gegenübersteht.

Ferner besteht bei der Ableitung der Anforderungen aus dem IST-System immer die Gefahr, dass alter Wein in neue Schläuche abgefüllt wird und mögliche neue, innovative Wege nicht erkannt werden.

7.5 Darstellung von Anforderungen

Die möglichen Darstellungsformen unterscheiden sich konzeptionell vor allem in zwei Aspekten:

- Art der Darstellung: *konstruktiv* oder *deskriptiv*
- *Formalitätsgrad* der Darstellung

Ein weiterer Freiheitsgrad besteht im *Detaillierungsgrad* der Darstellung.

7.5.1 Art der Darstellung

Deskriptive Darstellung

Eine deskriptive Darstellung betrachtet das zu spezifizierende System als schwarzen Kasten (Bild 7.7). Die Anforderungen werden durch Beschreibung der geforderten Zusammenhänge zwischen den geforderten Resultatdaten und den gelieferten Eingabedaten dargestellt.

BILD 7.7. Deskriptive Beschreibung von Anforderungen

Beispiele

- Darstellung mit Text in natürlicher Sprache:
  «Die Funktion Kontostand liefert den aktuellen Stand des Kontos für die eingegebene Kontonummer.»
- Darstellung in einer formalen Notation:
  Sqrt: Real → Real; Pre: $x \geq 0$; Post: $|Sqrt^2(x) - x| < \varepsilon \wedge \varepsilon \leq 10^{-16} \wedge \varepsilon \leq 10^{-6}x$.

Eine deskriptive Darstellung hat gewichtige Vorteile:

- Die Beschreibung beschränkt sich auf das äußere Verhalten (das heißt: wie das System mit seiner Umgebung interagiert); das ist es, worauf es eigentlich ankommt.
- Die Darstellung ist völlig lösungsneutral.

Diesen Vorteilen stehen jedoch ebenso gewichtige Nachteile gegenüber:

- Eine deskriptive Beschreibung real großer Systeme mit Text ist sehr umfangreich und wenig strukturiert. Die Zusammenhänge zwischen verschiedenen Anforderungen sind oft nicht erkennbar oder gar nicht dargestellt. Eine solche Darstellung ist daher fehlerträchtig und schwierig zu prüfen.
- Eine deskriptive Beschreibung real großer Systeme mit formalen Mitteln ist sehr schwierig und erfordert bestens ausgebildete Spezialisten. Die Prüfung einer solchen Darstellung auf ihre Adäquatheit stellt in vielen Fällen ein fast unüberwindbares Hindernis dar, weil die Vertreter des Kunden die Darstellung weder verstehen noch nachvollziehen können.

Deskriptive Darstellungen sind daher nur für die Darstellung der Anforderungen kleiner, überschaubarer Teilprobleme geeignet. Dort, vor allem bei der Modellierung echter Funktionen, ist ihr Einsatz sinnvoll.

Konstruktive Darstellung

Eine konstruktive Darstellung modelliert das zu spezifizierende System als eine Menge interagierender Komponenten (Bild 7.8). Die Anforderungen werden dargestellt, indem die Erzeugung der geforderten Resultate mit Hilfe der Komponenten und deren Zusammenarbeit in idealisierter Weise beschrieben wird.

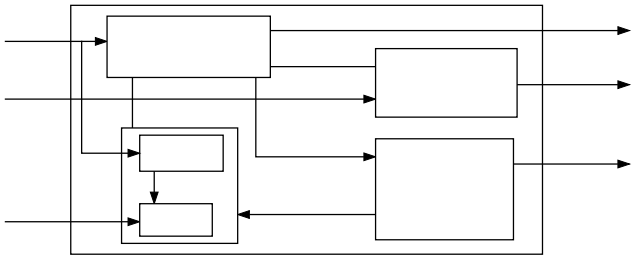


BILD 7.8. Konstruktive Beschreibung von Anforderungen

Beispiel: Das Datenflussdiagramm ist eine typische konstruktive Darstellung. Bild 7.9 zeigt die Spezifikation für die Aufbereitung und Anzeige eines Messwerts.

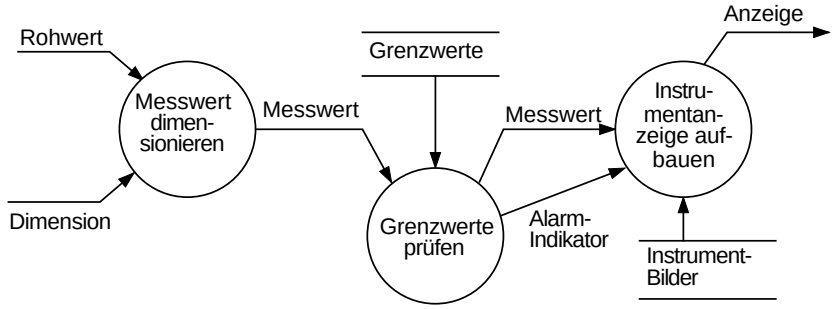


BILD 7.9. Datenflussdiagramm als typischer Vertreter einer konstruktiven Darstellung

Eine konstruktive Darstellung hat die folgenden Vorteile:

- Die Spezifikation ist ein anschauliches Modell der Problemstellung. Sie ist daher leicht verstehbar und nachvollziehbar.
- Die Darstellung ermöglicht die Zerlegung der Gesamtaufgabe in kleinere, besser überschaubare Teilaufgaben.
- Die Kombination von Teilen, die unterschiedlich stark formalisiert sind, ist möglich.
- Das Modell ist eine idealisierte Lösung. In vielen Fällen ist es daher möglich, die Lösung analog zur Struktur der Aufgabenstellung zu strukturieren. Dies spart Entwicklungsaufwand und vereinfacht das Verständnis der Lösungsstruktur.

Die Tatsache, dass das Modell eine idealisierte Lösung ist, stellt aber gleichzeitig auch einen Nachteil dar. Es besteht die Gefahr, dass

- bei der Modellierung auf die Implementierung geschielt wird, statt sich auf die Beschreibung der Anforderungen zu konzentrieren
- suboptimale Lösungen entstehen, wenn die Lösungsstruktur von der Anforderungsstruktur unverändert übernommen wird.

Konstruktive Darstellungen sind vor allem bei der Modellierung von Anforderungen im Großen angezeigt.

7.5.2 Formalitätsgrad der Darstellung

Bei jeder Systementwicklung müssen die vollständig informalen Ideen und Vorstellungen der Auftraggeber in eine vollständig formale Notation (nämlich den Programmcode) überführt werden. Verfahren zur Spezifikation von Anforderungen unterscheiden sich wesentlich in der Frage, wie weit die Formalisierung schon während des Spezifizierens erfolgen soll (Bild 7.10).

Eine *informale Spezifikation* erfolgt in der Regel deskriptiv mit natürlicher Sprache.

Formale Spezifikationen verwenden formale Kalküle, die sich mathematischer Mittel bedienen. Es sind sowohl konstruktive wie deskriptive Verfahren möglich. Petrinetze zum Beispiel sind konstruktiv; Spezifikation mit Logik ist deskriptiv.

Teilformale Spezifikationen basieren auf konstruktiven, anschaulichen Modellen, deren Konstruktionsregeln mit ihren Bedeutungen teilweise formalisiert sind, die aber auch nicht formale Elemente verwenden. Entity-Relationship-Modelle, Zustandsautomaten sowie die verschiedenen Arten der strukturierten und der objektorientierten Analyse sind typische Vertreter dieser Art von Spezifikation.

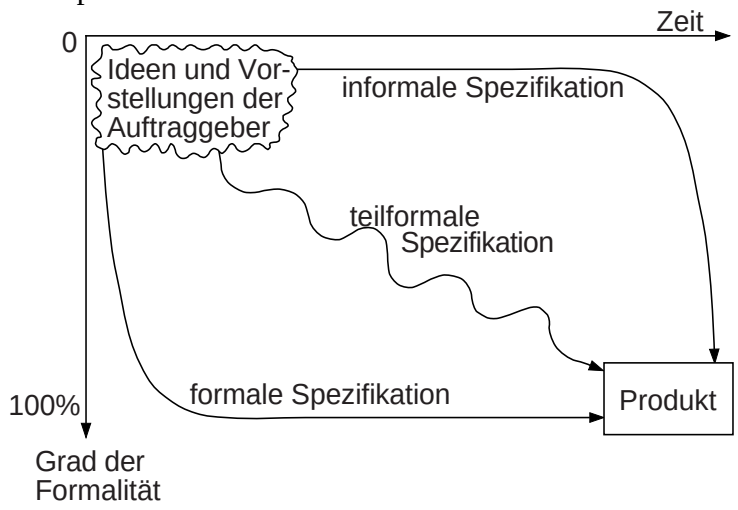


BILD 7.10. Verschiedene Wege bei der Formalisierung

Bei der Beurteilung der Vor- und Nachteile der drei Ansätze sind folgende Überlegungen maßgeblich:

- Informale Spezifikationen haben den großen Vorteil, leicht erstellbar zu sein. Jedoch enthalten große natürlichsprachige Spezifikationen in der Regel sehr viele Unklarheiten, Auslassungen, Mehrdeutigkeiten und Widersprüche. Diese werden beim Prüfen längst nicht alle entdeckt, da sorgfältiges Lesen die einzige zur Verfügung stehende Prüfmethode ist. Damit werden die Fehlerkosten zuwenig gesenkt und die Spezifikation wird unwirtschaftlich.

- Formale Spezifikationen sind immer eindeutig; die Widerspruchsfreiheit ist formal prüfbar. Jedoch ist die Erstellung einer vollständig formalen Spezifikation und ihre Prüfung (vor allem auf Adäquatheit und Vollständigkeit) in der Regel so aufwendig, dass die Kosten im Vergleich zum Nutzen zu hoch sind. Wiederum geht die Wirtschaftlichkeitsrechnung nicht auf.
- Teilformale Darstellungen weisen heute das beste Kosten/Nutzen-Verhältnis auf. Dies gilt insbesondere dann, wenn nicht alle Teile der Spezifikation den gleichen Grad der Formalisierung aufweisen müssen, sondern der Formalitätsgrad dem Entwicklungsrisiko, das mit den einzelnen Komponenten verbunden ist, angepasst werden kann. Teilformale Spezifikationen sind daher heute in den meisten Fällen das Mittel der Wahl.

Beispiel: Ausschnitte aus der Spezifikation der Steuerung eines Getränkeautomaten

Die nachfolgenden Bilder zeigen Beispiele einer informalen Spezifikation mit natürlicher Sprache (Bild 7.11), einer teilformalen und einer formalen Spezifikation (Bilder 7.12 und 7.13).

«Der Bediener drückt eine Wahltaste und bezahlt den geforderten Betrag. Sobald die Summe der eingeworfenen Münzen den geforderten Betrag übersteigt, wird das Getränk zubereitet und ausgegeben. Ferner wird das Wechselgeld berechnet und ausgegeben. Der Bedienvorgang endet, wenn das Getränk entnommen wird und wenn die Bedienung für länger als 45s unterbrochen wird. Mit einer Annulliertaste kann der Bedienvorgang jederzeit abgebrochen werden. Bereits eingeworfenes Geld wird beim Drücken der Annulliertaste zurückgegeben. Nach dem Drücken einer Wahltaste kann entweder bezahlt oder eine andere Wahltaste gedrückt werden. Die zuletzt getätigte Auswahl gilt.»	**Unklarheit:** Reihenfolge von Auswahl und Bezahlung? **Fehler:** Was ist bei Betragsgleichheit? **Mehrdeutigkeit:** Ist „und“ oder „oder“ gemeint? **Unklarheit:** Abbruch während der Ausgabe des Getränks? **Widerspruch:** Die oben geforderte Möglichkeit der Annullierung wird ausgeschlossen

BILD 7.11. Informale Spezifikation und deren Probleme

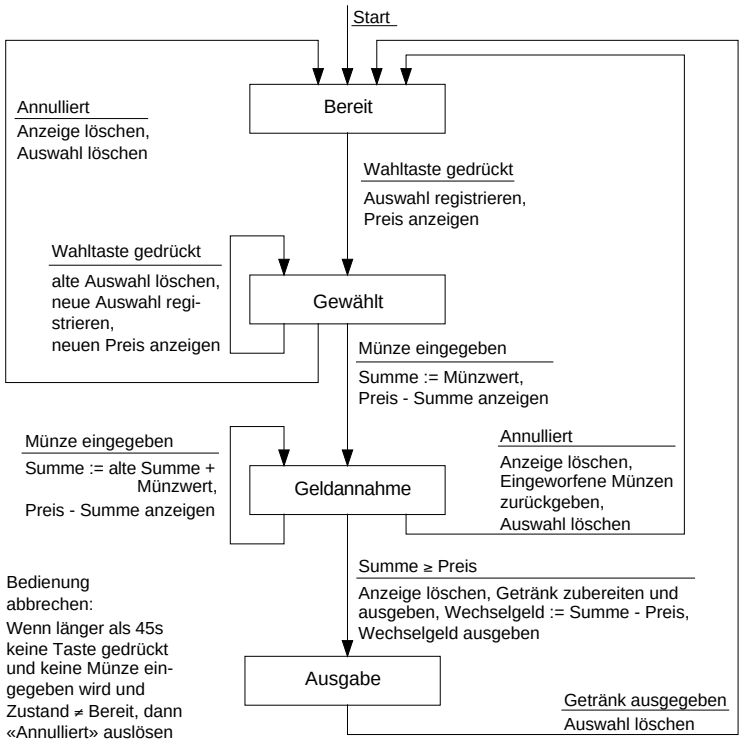


BILD 7.12. Teilformale Spezifikation der Steuerung eines Getränkeautomaten mit einem Zustandsautomaten

Sei m_i die i-te eingegebene Münze, $|m_i|$ der Wert dieser Münze, P der geforderte Preis und G die Funktion, welche die Getränkezubereitung auslöst. Dann muss gelten:

$\forall\, n \in \mathbb{N}\ \forall\, m_i,\ 1 \le i \le n$

$$\left[\sum_{i=1}^{n} |m_i| \ge P \ \wedge \ \sum_{i=1}^{n-1} |m_i| < P\right] \leftrightarrow \left[G(m_1 \circ m_2 \circ \ldots \circ m_n) \wedge \neg G(m_1 \circ m_2 \circ \ldots \circ m_{n-1})\right]$$

BILD 7.13. Formale Spezifikation des Zusammenhangs zwischen der Eingabe von Münzen und der Auslösung der Zubereitung in der Steuerung eines Getränkeautomaten

7.5.3 Detaillierungsgrad

Der Detaillierungsgrad der Darstellung ergibt sich aus der Abwägung folgender Punkte:

- *Kosten*: Je detaillierter die Anforderungen aufgeschrieben werden, desto teurer und zeitaufwendiger ist die Spezifikation.
- *Risiko*: Je weniger Details spezifiziert werden, desto höher ist das Risiko, dass das entwickelte System nicht den Wünschen und Bedürfnissen der Auftraggeber entspricht.
- *Entscheidungsspielraum*: Je detaillierter die Anforderungsspezifikation ausgeführt ist, desto weniger Spiel- und Gestaltungsraum bleibt für die Entwickler. Das mag einerseits beabsichtigt sein, um das Risiko zu senken, kann aber auch zu suboptimalen Ergebnissen führen, weil durch die engen Vorgaben kreative Lösungen verunmöglicht werden können.

7.6 Ausgewählte Spezifikationsmethoden und -Sprachen

7.6.1 Spezifikation mit natürlicher Sprache

Es gibt nur wenige spezifische Methoden für das Spezifizieren mit natürlicher Sprache; sie dienen vor allem zur Erkennung bzw. Vermeidung unvollständiger, mehrdeutiger und vager Sätze (Rupp 2002). Dabei wird beispielsweise auf folgende Punkte geachtet:

- Sätze mit vollständigen Satzstrukturen und im Aktiv schreiben,
- Nur Nomen verwenden, die in einem Glossar definiert sind,
- Für Verben, die Prozesse beschreiben, feste Bedeutungen festlegen,
- Nomen mit unspezifischer Bedeutung („der Kunde“, „die Anzeigedaten“, ...) hinterfragen und präzisieren,
- Nominalisierungen (zum Beispiel „die Initialisierung“) hinterfragen, weil sie unvollständig spezifizierte Prozesse verbergen können,
- Anforderungen immer als Hauptsätze formulieren und Nebensätze nur zur Vervollständigung der im Hauptsatz genannten Anforderung verwenden.

In der Regel sind die Anforderungen nummeriert. Ferner werden häufig feste Gliederungsschemata verwendet.

7.6.2 Algebraische Spezifikation

Algebraische Spezifikation ist eine deskriptive, formale Spezifikationsmethode. Sie wird vor allem für die formale Spezifikation komplexer Datentypen verwendet. Die Grundidee dabei ist, die Wirkung von Operationen auf den Objekten eines Typs durch die Angabe von Axiomen (Ausdrücken, die immer wahr sein müssen) festzulegen. Die Syntax des zu spezifizierenden Datentyps wird durch Angabe von Definitions- und Wertebereichen festgelegt. Das nachfolgen-

de Beispiel (Bild 7.14) zeigt die algebraische Spezifikation eines Stacks (Keller- oder Stapelspeicher, bei dem immer das zuletzt gespeicherte Element wieder entnommen werden kann).

Sei bool der Datentyp mit dem Wertebereich {false, true} und der Booleschen Algebra als Operationen. Sei ferner elem der Datentyp für die Datenelemente, die im spezifizierten Stack zu speichern sind.

```
TYPE Stack
FUNCTIONS
new:    ()            →   Stack;   -- neuen (leeren) Stack anlegen
push:   (Stack, elem) →   Stack;   -- Element hinzufügen
pop:    Stack         →   Stack;   -- zuletzt hinzugefügtes Element entfernen
top:    Stack         →   elem;    -- liefert zuletzt hinzugefügtes Element
empty:  Stack         →   bool;    -- wahr, wenn Stack kein Element enthält
full:   Stack         →   bool;    -- wahr, wenn Stack voll ist

AXIOMS
∀ s ∈ Stack, e ∈ elem
(1)  ¬ full(s) → pop(push(s,e)) = s          -- Pop hebt den Effekt von Push auf
(2)  ¬ full (s) → top(push(s,e)) = e         -- Top liefert das zuletzt gespeicherte Element
(3)  empty(new) = true                       -- ein neuer Stack ist leer
(4)  ¬ full (s) → empty(push(s,e)) = false   -- nach Push ist ein Stack nicht mehr leer
(5)  full(new) = false                       -- ein neuer Stack ist nicht voll
(6)  ¬ emtpy(s) → full(pop(s)) = false       -- nach Pop ist ein Stack niemals voll
```

BILD 7.14. Algebraische Spezifikation eines Stacks

7.6.3 Datenflussorientierte Spezifikation: Strukturierte Analyse

Strukturierte Analyse (DeMarco 1979, McMenamin und Palmer 1984, Yourdon 1989) ist eine teilformale, konstruktive Spezifikationsmethode. Die Anforderungen werden durch eine Hierarchie von *Datenflussdiagrammen* modelliert. Ein Datenflussdiagramm (Bild 7.9) besteht aus Aktivitäten, Datenflüssen und Speichern. Zur Modellierung der Systemumgebung werden außerdem Endknoten verwendet (Bilder 7.15, 7.16). Datenflussdiagramme basieren auf dem Prinzip der *datengesteuerten Verarbeitung*: jede Aktivität arbeitet dann, wenn die von ihr benötigten Datenflüsse eintreffen. Sie erzeugt bei ihrer Arbeit neue Datenflüsse. Diese steuern entweder andere Aktivitäten an oder verlassen das System als Ergebnis.

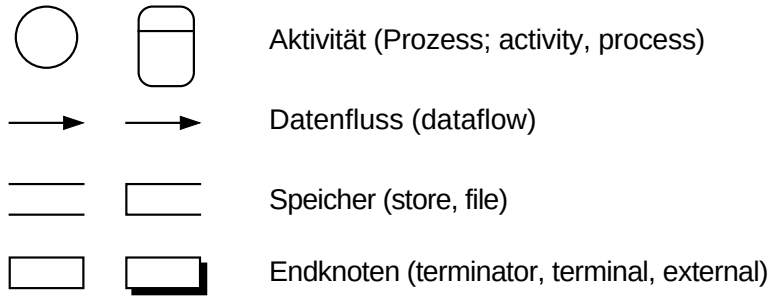


BILD 7.15. Notationen für Datenflussdiagramme

Ein Datenflussdiagramm wird wie folgt interpretiert:

- *Datenflüsse* transportieren Daten*pakete*, die von Aktivitäten oder Endknoten produziert bzw. konsumiert werden.

- *Aktivitäten* arbeiten nur dann, wenn alle von ihnen benötigten Eingabe-Datenflüsse vorliegen. Die Aktivität konsumiert die Daten, bearbeitet sie und produziert Ausgabe-Datenflüsse. Sie kann dabei zusätzlich Speicherinhalte lesen oder schreiben.
- *Speicher* modellieren Datenbehälter. Ihr Inhalt kann gelesen werden (ohne den Speicher zu verändern) und geschrieben werden (dabei wird der alte Inhalt zerstört).
- *Endknoten* sind Aktivitäten in der Systemumgebung.

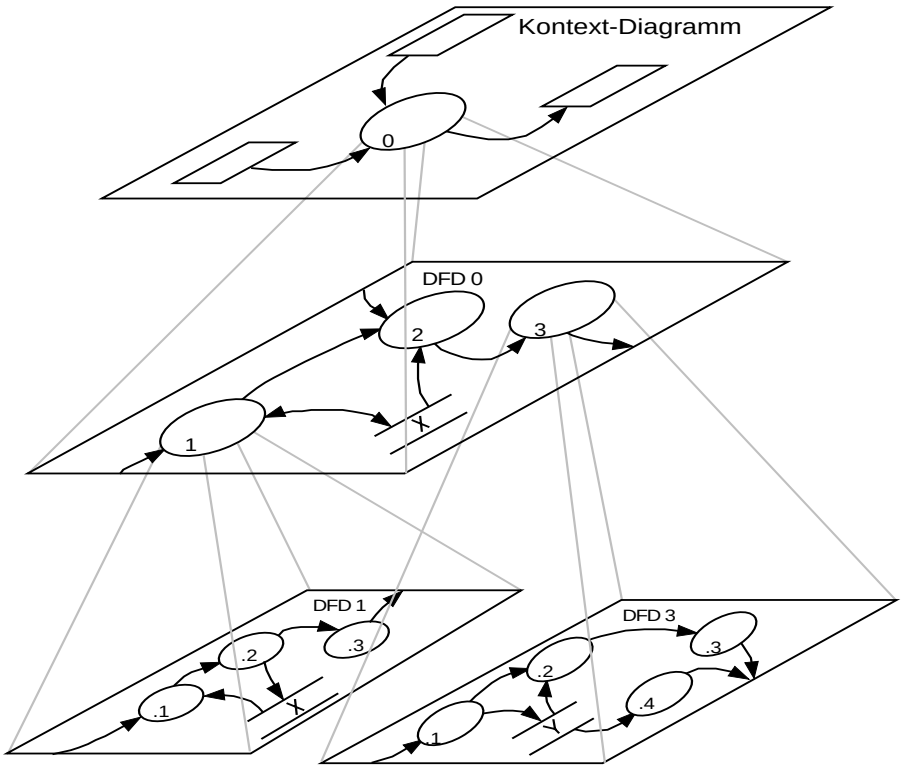


BILD 7.16. Beispiel einer DFD-Hierarchie in Strukturierter Analyse

Datenflussdiagramme sind Übersichtsmodelle. Zur Präzisierung müssen die Namen aller Datenflüsse und Speicher definiert werden Hierzu dient ein globales *Datenlexikon*, das alle verwendeten Datennamen definiert. Ferner muss die Funktionalität jeder nicht hierarchisch zerlegten Aktivität beschrieben werden. Hierzu dienen so genannte *Mini-Spezifikationen*, in denen der zu spezifizierende Zusammenhang zwischen Eingabe- und Ausgabedaten in freier Form beschrieben ist.

Die Datenflussdiagramme eines Systems können hierarchisch in Schichten angeordnet werden. Jede Ebene fasst die Datenflussdiagramme der darunter liegenden Ebenen zu je einer Aktivität zusammen. In der Regel wird ein dazu passendes hierarchisches Nummerierungsschema für Aktivitäten und DFD verwendet. Bei geeigneter Wahl der Zerlegung kann so ein komplexes Modell schrittweise in einfachere Teilmodelle zerlegt werden (Bild 7.16).

7.6.4 Verhaltensspezifikation mit Automaten

Verhaltensspezifikationen mit Automaten sind teilformale, konstruktive Modelle. Die Anforderungen werden als eine Menge von Zuständen, Zustandsübergängen, auslösenden Ereignissen und ausgelösten Aktionen beschrieben. Das Modell spezifiziert die Reaktionen des Systems auf eine gegebene Folge äußerer Ereignisse (Bild 7.12).

7.6.5 Objektorientierte Spezifikation

Objektorientierte Spezifikationsverfahren sind konstruktive, teilformale Methoden. Anfänglich gab es eine Vielzahl verschiedener Ansätze (zum Beispiel Booch (1994), Coad und Yourdon (1991a, b), Jacobson et al. (1992), Rumbaugh et al. (1991), Wirfs-Brock et al. (1990)). Diese sind ab etwa 1999 fast vollständig durch UML (Unified Modeling Language, Rumbaugh, Jacobson und Booch 1999) abgelöst worden. UML ist ein Industriestandard und derzeit die dominierende objektorientierte Modellierungssprache. Nachstehend werden die Prinzipien der objektorientierten Spezifikation beschrieben, wobei als Notation weitestgehend UML verwendet wird.

Die Grundidee ist, ein System durch eine Menge von *Objekten* zu spezifizieren, von denen jedes einen in sich geschlossenen Teil der Daten, der Funktionalität und des Verhaltens des Systems beschreibt. Dabei wird ein Ausschnitt der Realität auf Objekte und deren Eigenschaften abgebildet. Gleichartige Objekte werden durch *Klassen* modelliert.

DEFINITION 7.4. *Objekt (object).* Ein individuell erkennbares, von anderen Objekten eindeutig unterscheidbares Element der Realität.

Beispiel: Die konkrete Person Eva Müller, 36 Jahre alt, Dr. oec. publ., Leiterin Fertigung in der Firma AGP, verheiratet, ein Kind, ... wird als *Objekt* modelliert.

Objekteigenschaften werden als *Attribute* und *Attributwerte*, als *Beziehungen* zu anderen Objekten oder als *Operationen* auf den Objekten modelliert.

Beispiel: Das Objekt EVA MÜLLER hat das Attribut GESCHLECHT mit dem Wert WEIBLICH, die Beziehung LEITET zur ABTEILUNG FERTIGUNG und die Operation BEFÖRDERN.

DEFINITION 7.5. *Klasse (class).* Eine eindeutig benannte Einheit, welche eine Menge gleichartiger Objekte beschreibt.

Beispiel: Die Klasse MITARBEITER mit den Attributen NAME, VORNAME, GESCHLECHT, TITEL, ZIVILSTAND, ANZAHL KINDER, ... und den Beziehungen ARBEITET IN und LEITET zur Klasse ABTEILUNG beschreibt Personen der Art, wie Eva Müller eine ist.

Eine Klasse beschreibt den Aufbau, die Bearbeitungsmöglichkeiten und das mögliche Verhalten von Objekten dieser Klasse. Eine Klassendefinition besteht aus

- der Definition der Attribute der Klasse und der Wertebereiche dazu (lokale Merkmale)
- der Definition der Beziehungen zu anderen Klassen (referenzierte Merkmale)
- der Definition der Operationen, die auf Objekten der Klasse oder auf der Klasse selbst möglich sind

Dabei gibt es immer zwei Sichten für Klassen: In der *intensionalen* Sicht ist eine Klasse ein *Typ*. Alle Objekte der Klasse sind nach diesem Typ aufgebaut. In der *extensionalen* Sicht dagegen ist eine Klasse eine *Menge von Objekten*. Häufig werden beide Sichten miteinander vermischt.

Klassen stehen wie folgt in Beziehung zueinander:

- *Assoziation*: Die Objekte einer Klasse sind Merkmale von Objekten einer anderen Klasse. Gilt in der Regel auch umgekehrt, d.h. Assoziationen sind bidirektional.
- *Benutzung*: Die Objekte einer Klasse benutzen Attribute oder Operationen einer anderen Klasse zur Bereitstellung ihrer eigenen Attribute und Operationen.
- *Vererbung*: Eine Klasse ist Unterklasse einer anderen Klasse. Sie erbt alle Attribute und Operationen dieser Klasse, d.h. alle Objekte der Klasse verfügen über diese, ohne dass sie in der Klasse lokal definiert worden wären.
- Zu den Assoziationen werden *Kardinalitäten* modelliert: wie viele Objekte der assoziierten Klasse müssen mindestens / dürfen höchstens mit einem Objekt der eigenen Klasse assoziiert sein.

```
KLASSE Mitarbeiter im Monatslohn
UNTERKLASSE von Mitarbeiter
ATTRIBUTE (Name, Kardinalität, Wertebereich)
   Leistungslohnanteil      [1..1]    CHF
   Überzeitsaldo            [1..1]    Stunde
   Ferienguthaben           [1..1]    Tag
   ...
BEZIEHUNGEN (Name, Kardinalität, mit Klasse)
   eingestuft in            [1,1]     Lohnklasse
OPERATIONEN
   Lohn zahlen
      Voraussetzung: Mitarbeiter ist aktiv
      Ergebniszusicherung: Zahlungsauftrag zugunsten des Mitarbeiters ist erteilt mit Grundlohn aus
      Lohnklasse und Leistungslohnanteil
BENUTZT (Klasse.Operation)
   Zahlungsauftrag.Erteilen
```

BILD 7.17. Beispiel einer Klassendefinition am Beispiel einer Klasse aus einem Personalinformationssystem

Hinweis: Da MITARBEITER IM MONATSLOHN eine Unterklasse von MITARBEITER ist, werden alle Attribute (zum Beispiel NAME und VORNAME), Beziehungen und Operationen der Klasse MITARBEITER automatisch übernommen, ohne dass sie nochmals definiert werden müssten.

Es gibt heute eine Vielzahl verschiedener *Notationen* für Klassenmodelle. Die meisten Notationen verwenden Rechtecke für Klassen und Linien für Assoziationen. Die Klassensymbole werden häufig durch zwei Linien unterteilt: oben steht der Klassenname, in der Mitte die wichtigsten Attribute, unten die wichtigsten Operationen. Die Notation für Benutzung und Vererbung ist sehr uneinheitlich. Hier werden verwendet: Pfeile für Benutzung, gegabelte Linien für Vererbung. Für die Notation von Kardinalitäten gibt es eine verwirrende Fülle von graphischen und numerischen Notationssystemen. In Bild 7.18 wird die Notation der UML (Unified Modeling Language) verwendet. Die Kardinalitäten sind in Beziehungsrichtung zu lesen, zum Beispiel „Eine ABTEILUNG BESCHÄFTIGT mindestens einen und höchstens beliebig viele MITARBEITER“ (ein Stern steht für beliebig viele).

Als *Alternative* oder als *Ergänzung* zu Klassenmodellen können *Objektmodelle* verwendet werden. In solchen Modellen werden keine konkreten Objekte (d.h. solche mit bekannter Identität und bekannten Attributwerten) modelliert, sondern *abstrakte Objekte*, die als Muster bzw. als Repräsentanten für konkrete Objekte stehen. In den meisten heute verwendeten Ansätzen werden Objektmodelle entweder *gar nicht verwendet* oder sie dienen in *Ergänzung* zu einem Klassenmodell zur Modellierung des Arbeitskontextes der Objekte einer Klasse. Objektmodelle *anstelle* von Klassenmodellen haben große Vorteile, wenn verschiedene Objekte der gleichen Klasse zu modellieren sind, und wenn ein Modell hierarchisch in Komponenten zerlegt werden soll. Dagegen ist die Modellierung von Assoziationen und Vererbungsbeziehungen in Objektmodellen umständlicher als in Klassenmodellen.

Bild 7.23 zeigt eine Situation, in der ein Objektmodell einem Klassenmodell in der Ausdruckskraft überlegen ist. In manchen Klassenmodellen, so auch in UML, behilft man sich, indem die Beziehung zusätzlich mit den *Rollen*, welche die in Beziehung gesetzten Objekte spielen (d.h. hier TUTOR und ERSTSEMESTER) beschriftet werden.

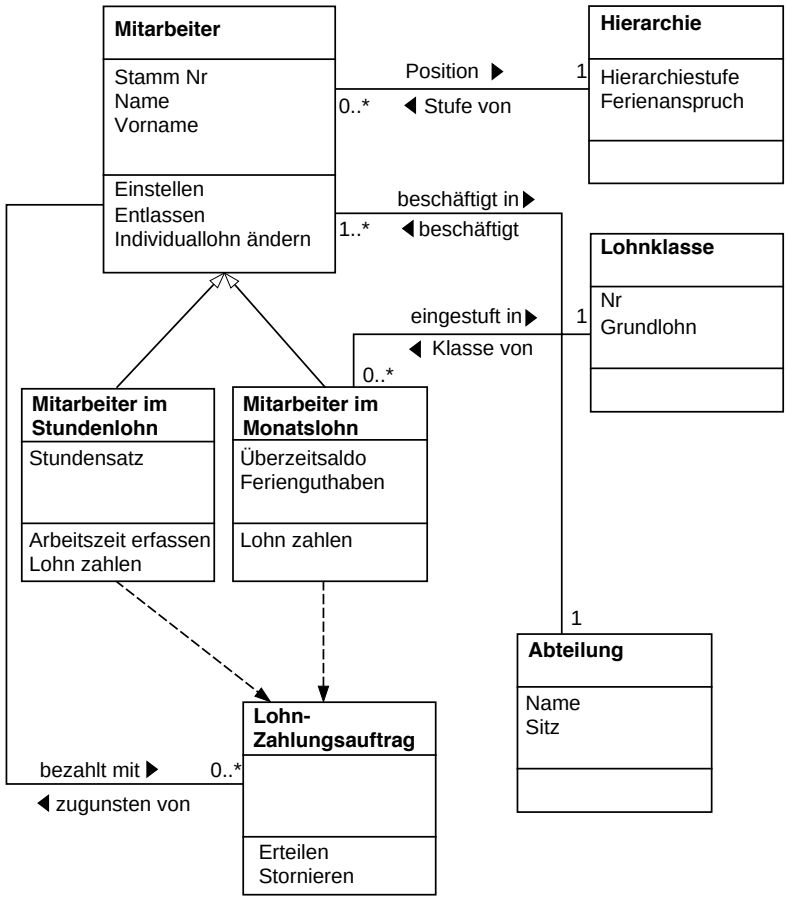


BILD 7.18. Beispiel eines Klassenmodells (stark vereinfachtes Modell eines Personalinformationssystems)

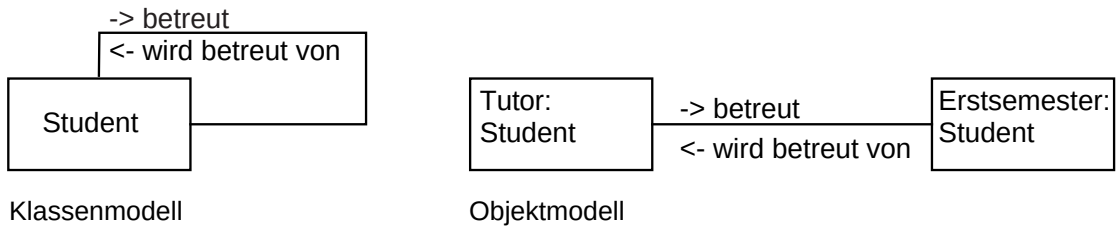


BILD 7.19. Klassenmodell vs. Objektmodell

7.6.6 Spezifikation mit Anwendungsfällen

Die Klassenmodelle in Objektorientierten Spezifikationen modellieren weder den Systemkontext noch die Interaktionen zwischen einem System und seiner Umgebung. Weil dies aber beides wichtige Elemente einer Anforderungsspezifikation sind, werden objektorientierte Spezifikationen heute meist durch ein *Anwendungsfallmodell*, welches die Interaktionen zwischen den systemexternen *Akteuren* und dem System modelliert, ergänzt (Jacobson et al. 1992).

Definition 7.6. *Anwendungsfall (use case)*. Eine durch genau einen Akteur angestoßene Folge von Systemereignissen, welche für den Akteur ein Ergebnis produziert und an welchem weitere Akteure teilnehmen können.

Jeder Anwendungsfall spezifiziert die notwendige Folge von Interaktionschritten bei der Ausführung einer essentiellen Systemfunktion. Er wird informal durch Text oder teilformal,

beispielsweise durch Zustandsautomaten (vgl. Bild 7.12), modelliert. Ein Anwendungsfalldiagramm zeigt den Systemkontext und die Menge aller Anwendungsfälle (Bild 7.24). Die zwei «uses»-Beziehungen modellieren die gemeinsame Verwendung eines Sub-Anwendungsfalls durch zwei andere Anwendungsfälle.

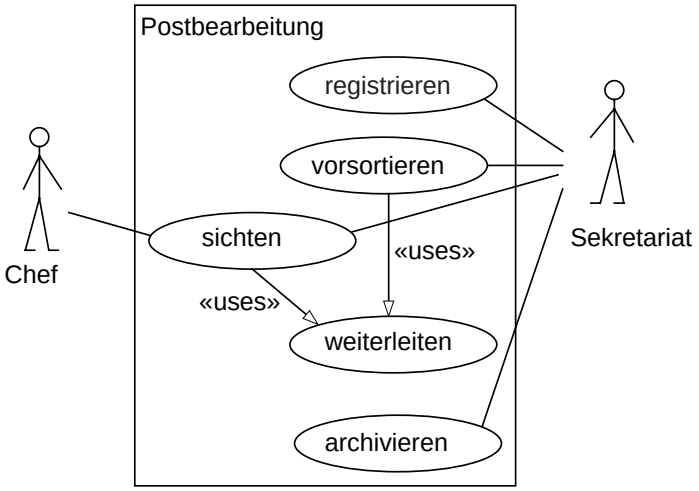


BILD 7.24. Anwendungsfalldiagramm

7.7 Prüfung von Anforderungen

7.7.1 Worum es geht

Beim Prüfen einer Anforderungsspezifikation geht es darum, Abweichungen von der geforderten Qualität der Spezifikation festzustellen. Mit anderen Worten, bei der Prüfung sollen möglichst viele Fehler, Lücken, Unklarheiten, Mehrdeutigkeiten, etc. gefunden (und anschließend behoben) werden.

Dabei haben nicht alle Qualitätsmerkmale das gleiche Gewicht. In erster Priorität sollte auf Adäquatheit, Vollständigkeit, Widerspruchsfreiheit und Verständlichkeit geprüft werden, in zweiter Priorität auf Prüfbarkeit und Eindeutigkeit und in dritter Priorität auf alle übrigen Merkmale.

Die Prüfung einer Spezifikation auf Adäquatheit, Vollständigkeit und Widerspruchsfreiheit wird auch *Validierung* genannt.

7.7.2 Beteiligte

Die Prüfung erfolgt sinnvollerweise unter Federführung von Informatikern, die insbesondere die verwendete Darstellung ohne fremde Hilfe interpretieren können. Zusätzlich müssen aber die Vertreter der übrigen Beteiligten zwingend in den Prüfprozess einbezogen werden, denn nur sie können schlussendlich die Adäquatheit und Vollständigkeit der Spezifikation beurteilen.

7.7.3 Zeitpunkt der Prüfung

Die abschließende Prüfung einer Spezifikation erfolgt zu einem Zeitpunkt, wo

- die Spezifikation fertig ist
- aber noch genug Zeit bleibt, die gefundenen Mängel zu beheben.

Bei umfangreichen Spezifikationen sind zusätzlich Zwischenprüfungen begleitend zur Erstellung der Spezifikation erforderlich.

7.7.4 Prüfverfahren

Für die Prüfung einer Anforderungsspezifikation kommen vier Verfahren in Betracht

- Reviews
- Prüf- und Analysemittel in Werkzeugen
- Simulation
- Prototypen.

Ein *Review* ist eine formell organisierte Zusammenkunft von Personen zur inhaltlichen oder formellen Überprüfung eines Produktteils (Dokument, Programmstück, etc.) nach vorgegebenen Prüfkriterien (vgl. Kapitel 9.4). Reviews sind *das* Mittel zur Prüfung von Dokumenten.

Prüf- und Analysemittel in Werkzeugen werden immer dann eingesetzt, wenn eine Spezifikation mit Hilfe von Werkzeugen erstellt wurde. Insbesondere Lücken und Widersprüche in der Spezifikation lassen sich mit solchen Prüfverfahren finden. Beispielsweise kann ein Werkzeug prüfen, ob jeder verwendete Datenname auch irgendwo definiert ist.

Mit einer *Simulation* kann die Adäquatheit des Verhaltens des spezifizierten Systems in bestimmten Situationen untersucht werden.

Ein *Prototyp* ist ein lauffähiges Stück Software, welches Teile eines zu entwickelnden Systems vorab realisiert (vgl. Kapitel 3.2.5). Er dient als Modell für die weitere Entwicklung oder für das zu schaffende Produkt. Ein Prototyp ist das mächtigste verfügbare Mittel für die Beurteilung der Adäquatheit einer Spezifikation durch die zukünftigen Benutzer. Er ermöglicht in beschränktem Rahmen eine Erprobung des gewünschten Systems in seiner geplanten Einsatzumgebung. Aufgrund der im Vergleich zu anderen Prüfverfahren hohen Kosten für einen Prototyp muss in jedem Einzelfall entschieden werden, wie weit die Entwicklung von Prototypen für die Prüfung von Anforderungen aufgrund des Entwicklungsrisikos notwendig ist.

7.8 Verwalten von Anforderungen

Anforderungen können sich im Verlauf der Entwicklung ändern, was eine Anpassung der bestehenden Anforderungsspezifikation erfordert. Andererseits ist ohne eine stabile Spezifikation keine geordnete und zielgerichtete Entwicklung möglich. Verwalten von Anforderungen (*requirements management*), zielt daher darauf ab, die Anforderungen grundsätzlich stabil zu halten, aber dennoch Veränderungen kontrolliert zuzulassen.

Hierzu wird zunächst einmal eine Konfigurationsverwaltung für Anforderungen (vgl. Kapitel 11) benötigt. Ferner müssen die Anforderungen *verfolgbar* sein und zwar sowohl rückwärts (wo kommt eine Anforderung her?), als auch vorwärts (wo ist welche Anforderung in den Entwurf bzw. die Implementierung eingegangen?). Ohne eine solche Verfolgbarkeit ist es bei geplanten Änderungen von Anforderungen sehr schwierig zu beurteilen, welche Auswirkungen eine solche Änderung nach sich zieht.

Aufgaben

7.1 Welches sind die Eigenschaften einer guten Spezifikation? Begründen Sie ihre Aussagen.

7.2 Nehmen Sie an, Sie bräuchten ein Zutrittskontrollsystem (Schlüsselkarten o.ä.) für die Haustür eines Bürogebäudes. Beschreiben Sie ihre Vorstellungen über das Verhalten dieses Systems mit Szenarien.

7.3 Spezifizieren Sie einen Taschenrechner Ihrer Wahl in natürlicher Sprache. Gliedern Sie die Anforderungen gemäß der Struktur der „Anforderungen an das Produkt“ in Bild 7.5.

Ergänzende und vertiefende Literatur

Davis (1993) ist ein Lehrbuch über Requirements Engineering, das vor allem die klassischen Techniken beschreibt.

Gause und Weinberg (1989) beleuchten primär den Aspekt der Anforderungsgewinnung und der sozialen Dimension des Requirements Engineerings.

Kotonya und Sommerville (1998) ist eine Monographie über Requirements Engineering.

Oestereich (1998) gibt eine sehr gute Einführung in die UML.

Verschiedene objektorientierte Spezifikationsverfahren sind beschrieben in: Booch (1994), Rumbaugh, Jacobson und Booch (1999), Coad und Yourdon (1991a), Jacobson et al. (1991), Rumbaugh et al. (1991).

Die folgenden Autoren behandeln klassische Spezifikationsverfahren. Information Engineering: Martin (1990); SSADM: Ashworth und Goodland (1990); Strukturierte Analyse: DeMarco (1979), McMenamin und Palmer (1984) sowie Yourdon (1989).

Rupp (2002) behandelt das systematische Erstellen von Anforderungsspezifikationen mit natürlicher Sprache.

McDermid (1991) beschreibt verschiedene, teilweise weniger bekannte klassische Spezifikationsverfahren sowie Verfahren zur formalen Spezifikation (Z, VDM, algebraische Spezifikation).

Jackson (1995) bietet in Form einer Sammlung kurzer Essays eine Fülle vertiefter Einsichten über Requirements Engineering.

Pepper et al. (1982) geben eine Einführung in die algebraische Spezifikation.

Zitierte Literatur

Ashworth, C., M. Goodland (1990). *SSADM: A Practical Approach*. London: McGraw-Hill.

Boehm, B. (1981). *Software Engineering Economics*. Englewood Cliffs, N.J.: Prentice-Hall.

Booch, G. (1994). *Object Oriented Analysis and Design with Applications*. Second Edition. Redwood City, Ca.: Benjamin/Cummings.

Coad, P., E. Yourdon (1991a). *Object-Oriented Analysis*. 2nd edition. Englewood Cliffs, N.J.: Prentice-Hall. [auf Deutsch: *Objektorientierte Analyse*, 1994 im gleichen Verlag]

Coad, P., E. Yourdon (1991b). *Object-Oriented Design*. Englewood Cliffs, N.J.: Prentice-Hall. [auf Deutsch: *Objektorientiertes Design*, 1994 im gleichen Verlag]

Davis, A.M. (1993). *Software Requirements: Objects, Functions and States*. Englewood-Cliffs, N.J.: Prentice-Hall.

DeMarco, T. (1979). *Structured Analysis and System Specification*. Englewood Cliffs, N.J.: Prentice-Hall (Yourdon Press).

Gause, D.C., G.M. Weinberg (1989). *Exploring Requirements: Quality before Design*. New York: Dorset House. [auf Deutsch: *Software Requirements: Anforderungen erkennen, verstehen und erfüllen*. München: Hanser, 1993.]

IEEE (1990). *Standard Glossary of Software Engineering Terminology*. IEEE Std 610.12-1990. IEEE Computer Society Press.

IEEE (1993). *IEEE Recommended Practice for Software Requirements Specifications*. IEEE Standard 830-1993. IEEE Computer Society Press.

Jackson, M.A. (1995). *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices*. Wokingham: Addison-Wesley (ACM Press books).

Jacobson, I., M. Christerson, P. Jonsson, G. Övergaard (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Amsterdam; Reading, Mass.: Addison-Wesley.

Kotonya, G., I. Sommerville (1998). *Requirements Engineering: Processes and Techniques*. Chichester, etc.: John Wiley & Sons.

Martin, J. (1990). *Information Engineering*. *Book 2: Planning and Analysis*. Englewood Cliffs, N.J.: Prentice-Hall.

McDermid, J. (1991). *Software Engineer's Reference Book*. Oxford: Butterworth-Heinemann.

McMenamin, S.M., J.F. Palmer (1984). *Essential Systems Analysis*. New York: Yourdon Press.

Oestereich, B. (1998). *Objektorientierte Softwareentwicklung*. München: Oldenbourg.

Pepper, P. et al. (1982). Abstrakte Datentypen: Die algebraische Spezifikation von Rechenstrukturen. *Informatik-Spektrum* **5**, (1982). 107-119.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, W. Lorensen (1991). *Object-Oriented Modeling and Design*. Englewood Cliffs, N.J.: Prentice-Hall.
[auf Deutsch: *Objektorientiertes Modellieren und Entwerfen*, München: Hanser, 1993]

Rumbaugh, J., Jacobson, I., Booch, G. (1999). *The Unified Modeling Language Reference Manual*. Reading, Mass.: Addison-Wesley.

Rupp, C. (2002). *Requirements-Engineering und -Management*. 2. Auflage. München: Hanser.

Wirfs-Brock, R., B. Wilkerson, L. Wiener (1990). *Designing Object-Oriented Software*. Englewood Cliffs, N.J.: Prentice-Hall.
[auf Deutsch: *Objektorientiertes Software Design*, München: Hanser, 1993]

Yourdon, E. (1989). *Modern Structured Analysis*. Englewood Cliffs, N.J.: Prentice-Hall.