Helmut Balzert

Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering

3. Auflage

Unter Mitwirkung von
Heide Balzert
Rainer Koschke
Uwe Lämmel
Peter Liggesmeyer
Jochen Quante

Spektrum
AKADEMISCHER VERLAG

7 Fallstudie: SemOrg – Die Spezifikation

Um die verschiedenen Techniken und Konzepte zu veranschaulichen, wird eine **kaufmännische Fallstudie SemOrg** (für Seminarorganisation) durchgängig verwendet und referenziert. Das folgende **Lastenheft** und **Pflichtenheft** verwendet nur natürlichsprachliche Anforderungen zur Festlegung der Anforderungen. Der Aufbau von Lasten- und Pflichtenheft ist im Kapitel »Schablonen für Lastenheft, Pflichtenheft und Glossar«, S. 492, erklärt. Fachbegriffe werden in einem **Glossar** definiert (siehe unten). TBD steht für *To be definied* – muss noch definiert werden bzw. *To be determined* – muss noch festgelegt werden.

Lastenheft SemOrg

Version	Autor	Quelle	Status	Datum	Kommentar
0.1	Manfred Mustermann	Geschäftsführer Teachware	in Bearbeitung	10/09	

Voreinstellungen *(Kursiv dargestellt)*:
Priorität aus Auftraggebersicht = {hoch, *mittel*, niedrig}
Priorität aus Auftragnehmersicht = {hoch, *mittel*, niedrig}
Stabilität der Anforderung = {fest, *gefestigt*, volatil}
Kritikalität der Anforderung = {hoch, mittel, *niedrig*, keine}
Entwicklungsrisiko der Anforderung = {hoch, mittel, *niedrig*}

1 Visionen und Ziele

/LV10/ Die Firma Teachware soll durch das System in die Lage versetzt werden, die von ihr veranstalteten Seminare sowie Kunden und Dozenten effizient rechnerunterstützt zu verwalten.
/LV20/ Die Kunden der Firma Teachware sollen über das Web möglichst viele Vorgänge selbst durchführen können.
/LZ10/ Ein Interessent oder ein Kunde kann mindestens 20 Stunden jeden Tag Seminare und Veranstaltungen über das Web selektieren und eine Veranstaltung online buchen, damit die Mitarbeiter der Fa. Teachware von solchen Tätigkeiten entlastet werden.

H. Balzert, *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*,
DOI 10.1007/978-3-8274-2247-7_7, © Spektrum Akademischer Verlag Heidelberg 2009

2 Rahmenbedingungen

/LR10/ SemOrg ist eine kaufmännisch/administrative Web-Anwendung.
/LR20/ Zielgruppe sind die Mitarbeiter der Fa. Teachware (Kundensachbearbeiter, Seminarsachbearbeiter, Veranstaltungsbetreuer) sowie Interessenten und Kunden.

3 Kontext und Überblick

/LK10/ Das System besitzt eine Softwareschnittstelle zu einem Buchhaltungssystem.
/LK20/ Das System ist mindestens 20 Stunden pro Tag im Intranet der Fa. Teachware und im Internet verfügbar.

4 Funktionale Anforderungen

/LF10/ Das System *soll* Interessenten und Kunden die Möglichkeit bieten, sich über Seminare und Veranstaltungen zu informieren, Veranstaltungen zu buchen und einen Seminarkatalog anzufordern.
/LF20/ Das System *muss* dem Kundensachbearbeiter die Möglichkeit bieten, neue Kunden/Firmen zu erfassen und vorhandene Kunden-/Firmendaten zu aktualisieren und Kunden/Firmen zu löschen.
/LF30/ Das System *muss* dem Kundensachbearbeiter die Möglichkeit bieten, Seminare und Veranstaltungen zu selektieren, Veranstaltungen für Interessenten und Kunden zu buchen und für angeforderte Seminarkataloge Versandpapiere zu erstellen.
/LF40/ Das System *muss* dem Seminarsachbearbeiter die Möglichkeit bieten, neue Dozenten zu erfassen und vorhandene Dozentendaten zu aktualisieren und Dozenten zu löschen.
/LF50/ Das System *muss* dem Seminarsachbearbeiter die Möglichkeit bieten, neue Seminare und Veranstaltungen zu erfassen, vorhandene zu modifizieren oder zu löschen.
/LF60/ Das System *soll* dem Seminarsachbearbeiter die Möglichkeit bieten, für alle Veranstaltungen Hotels auszuwählen und Räume zu reservieren.
/LF70/ Das System *muss* Kunden-, Firmen-, Seminar-, Veranstaltungs- und Dozentendaten permanent speichern.
/LF80/ Das System *muss* fähig sein, dem Buchhaltungssystem Rechnungsdatensätze mindestens einmal am Tag zur Verfügung zu stellen.

5 Qualitätsanforderungen

/LQE10/ Alle Reaktionszeiten auf Benutzeraktionen müssen unter 5 Sekunden liegen.

Systemqualität	sehr gut	gut	normal	nicht relevant
Funktionalität		X		
Zuverlässigkeit			X	
Benutzbarkeit		X		
Effizienz			X	
Wartbarkeit			X	
Portabilität				X

Tab. 7.0-1: Qualitätsanforderungen an SemOrg.

Glossar SemOrg

Version	Autor	Quelle	Status	Datum	Kommentar
0.1	Manfred Mustermann	Geschäftsführer Teachware	in Bearbeitung	10/09	

Dozent: Führt als freier Mitarbeiter eine oder mehrere angebotene →Veranstaltungen durch. Ist fachlich in der Lage, ein oder mehrere →Seminare abzuhalten.
Firma: Mitarbeiter einer Firma (Ansprechpartner), der für die Aus- und Weiterbildung von Mitarbeitern zuständig ist und sich über Dienstleistungen informiert oder Mitarbeiter zu öffentlichen →Veranstaltungen schickt oder firmeninterne Veranstaltungen bucht.
Interessent: →Kunde, der sich für Dienstleistungen, z. B. Seminarkatalog, interessiert, aber noch an keiner →Veranstaltung teilgenommen hat.
Kunde: Mitarbeiter einer Firma oder Privatperson, der bzw. die an Dienstleistungen interessiert ist, oder ein Seminar bucht und besucht (→Teilnehmer, →Interessent).
Kundensachbearbeiter: Verantwortlich für die Kommunikation mit →Kunden und →Firmen einschließlich der Auskunftserteilung und Buchung.
Seminar: →Seminartyp.
Seminarsachbearbeiter: Verantwortlich für die Planung und Terminierung von →Seminaren und →Veranstaltungen. Zuständig für die Kommunikation und Akquirierung von →Dozenten.
Seminartyp: Beschreibt die Gemeinsamkeiten, die eine Menge von →Veranstaltungen besitzen wie Titel, Zielsetzung, Inhalt, Voraussetzungen.
Seminarveranstaltung: →Veranstaltung.
Teilnehmer: →Kunde, der an einer →Veranstaltung teilnimmt bzw. teilgenommen hat.
Veranstaltung: →Seminar, das zu einem festgelegten Zeitpunkt, an einem festgelegten Ort von einem oder mehreren →Dozenten durchgeführt wird.

Veranstaltungsbetreuer: Betreut die →Teilnehmer und →Dozenten einer →Veranstaltung.

Pflichtenheft SemOrg

Version	Autor	Quelle	Status	Datum	Kommentar
0.1	Manfred Muster-mann	Geschäfts-führer Teachware	in Bear-beitung	11/09	Verfeine-rung des Lastenhefts V0.1

Voreinstellungen *(Kursiv dargestellt)*:
Priorität aus Auftraggebersicht = {hoch, *mittel*, niedrig}
Priorität aus Auftragnehmersicht = {hoch, *mittel*, niedrig}
Stabilität der Anforderung = {fest, *gefestigt*, volatil}
Kritikalität der Anforderung = {hoch, mittel, *niedrig*, keine}
Entwicklungsrisiko der Anforderung = {hoch, mittel, *niedrig*}

Hinweis

Alle Anforderungen im Pflichtenheft, die einen Bezug zu einer Anforderung im Lastenheft haben, müssen eine entsprechende Referenz auf das Lastenheft haben. Da das Pflichtenheft eine Verfeinerung und Erweiterung des Lastenhefts darstellt, gibt es neue Anforderungen, von denen kein Bezug zum Lastenheft hergestellt werden kann. Es gibt dann keine entsprechende Referenz.

1 Visionen und Ziele

/V10/ (/LV10/) Die Firma Teachware soll durch das System in die Lage versetzt werden, die von ihr veranstalteten Seminare sowie Kunden und Dozenten effizient rechnerunterstützt zu verwalten.
/V20/ (/LV20/) Die Kunden der Firma Teachware sollen über das Web möglichst viele Vorgänge selbst durchführen können.
/Z10/ (/LZ10/) Ein Interessent oder ein Kunde kann mindestens 20 Stunden jeden Tag Seminare und Veranstaltungen über das Web selektieren und eine Veranstaltung online buchen, damit die Mitarbeiter der Fa. Teachware von solchen Tätigkeiten entlastet werden.

2 Rahmenbedingungen

/R10/ (/LR10/) SemOrg ist eine kaufmännisch/administrative Web-Anwendung.
/R20/ (/LR20/) Zielgruppe sind die Mitarbeiter der Fa. Teachware (Kundensachbearbeiter, Seminarsachbearbeiter, Veranstaltungsbetreuer) sowie Interessenten und Kunden.
/R30/ Das System wird in einer Büroumgebung eingesetzt.
/R40/ (/LZ10/) Die tägliche Betriebszeit des Systems muss mindestens 20 Stunden jeden Tag betragen.

/R50/ Der Betrieb des Systems muss unbeaufsichtigt ablaufen.
/R60/ Eingesetzte Software auf der Zielmaschine: Client: Webbrowser (Die marktführenden 3 Webbrowser müssen unterstützt werden), Server: Betriebssystem Windows.
/R70/ Hardwarevoraussetzungen: Client: PC, Bildschirm mit mindestens XGA-Auflösung (1024 x 768), Server: TBD.
/R80/ (/LK10/) Netzwerkverbindung des Servers zum Buchhaltungsrechner.
/R90/ (/LK20/) Alle Clients sind über ein Intranet mit dem Server verbunden, der Server hat einen Internetanschluss.
/R100/ Die Entwicklungsumgebung kann identisch mit der Zielumgebung sein.

3 Kontext und Überblick

/K10/ (/LK10/) Das System besitzt eine Softwareschnittstelle zu einem Buchhaltungssystem: TBD.

4 Funktionale Anforderungen

/F10/ (/LF10/) Das System *soll* Interessenten und Kunden die Möglichkeit bieten, sich über Seminare und Veranstaltungen zu informieren, Veranstaltungen zu buchen und einen Seminarkatalog anzufordern.
/F11/ Wenn ein Kunde oder eine Firma sich von einer bereits gebuchten Veranstaltung mehr als X Wochen vor der Veranstaltung abmeldet, dann *muss* das System Stornogebühren in Höhe von Y Euro berechnen oder nach einem Ersatzteilnehmer fragen.
/F12/ Wenn ein Kunde oder eine Firma sich von einer bereits gebuchten Veranstaltung später als X Wochen vor der Veranstaltung abmeldet, dann *muss* das System Stornogebühren in Höhe der Veranstaltungsgebühr berechnen oder nach einem Ersatzteilnehmer fragen.
/F20/ (/LF20/) Das System *muss* dem Kundensachbearbeiter die Möglichkeit bieten, neue Kunden/Firmen zu erfassen und vorhandene Kunden-/Firmendaten zu aktualisieren und Kunden/Firmen zu löschen.
/F30/ (/LF30/) Das System *muss* dem Kundensachbearbeiter die Möglichkeit bieten, Seminare und Veranstaltungen zu selektieren, Veranstaltungen für Interessenten und Kunden zu buchen und zu stornieren sowie für angeforderte Seminarkataloge Versandpapiere zu erstellen.
/F40/ (/LF40/) Das System *muss* dem Seminarsachbearbeiter die Möglichkeit bieten, neue Dozenten zu erfassen, vorhandene Dozentendaten zu aktualisieren, Dozenten zu löschen und Dozenten Seminaren und Veranstaltungen zuzuordnen.

/F50/ (/LF50/) Das System *muss* dem Seminarsachbearbeiter die Möglichkeit bieten, neue Seminare und Veranstaltungen zu erfassen, vorhandene zu modifizieren oder zu löschen und Veranstaltungen Seminaren zuzuordnen.
/F60/ (/LF60/) Das System *soll* dem Seminarsachbearbeiter die Möglichkeit bieten, für alle Veranstaltungen Hotels auszuwählen und Räume zu reservieren.
/F61/ Das System *muss* dem Seminarsachbearbeiter und dem Kundensachbearbeiter die Möglichkeit bieten, Veranstaltungen zu stornieren, die angemeldeten Teilnehmer zu informieren, ihnen alternative Veranstaltungen anzubieten oder bei einem Dozentenausfall nach alternativen Dozenten zu suchen und dem Seminarsachbearbeiter vorzuschlagen.
/F70/ (/LF70/) Das System *muss* folgende Kundendaten (maximal 50.000) permanent speichern: Kunden-Nr., Name, Adresse, Kommunikationsdaten, Geburtsdatum, Funktion, Umsatz, Kurzmitteilung, Notizen, Info-Material, Kunde seit.
/F80/ (/LF70/) Das System *muss* folgende Firmendaten (maximal 10.000) permanent speichern, wenn ein Kunde zu einer Firma gehört: Firmenkurzname, Firmenname, Adresse, Kommunikationsdaten, Ansprechpartner, Abteilung, Geburtsdatum, Funktion des Ansprechpartners, Kurzmitteilung, Notizen, Umsatz, Kunde seit.
/F90/ (/LF70/) Das System *muss* folgende Seminartypdaten (maximal 10.000) permanent speichern: Seminarkurztitel, Seminartitel, Zielsetzung, Methodik, Inhaltsübersicht, Tagesablauf, Dauer, Unterlagen, Zielgruppe, Voraussetzungen, Gebühr ohne MWST, max. Teilnehmerzahl, min. Teilnehmerzahl.
/F100/ (/LF70/) Das System *muss* folgende Veranstaltungsdaten (maximal 100.000) permanent speichern: Veranstaltungs-Nr., Dauer (in Tagen), Vom, Bis, Tagesraster-Anfang, Tagesraster-Ende, Anfang erster Tag, Ende letzter Tag, Veranstaltungsort (Hotel/Firma, Adresse, Raum), Kooperationspartner, Öffentlich (Ja/Nein), Netto-Preis, Stornogebühr, min. Teilnehmerzahl, max. Teilnehmerzahl, Teilnehmer aktuell, Durchgeführt (Ja/Nein).
/F110/ (/LF70/) Das System *muss* folgende Dozentendaten (maximal 5.000) permanent speichern: Dozenten-Nr., Name, Adresse, Kommunikationsdaten, Geburtsdatum, Biografie, Honorar pro Tag, Kurzmitteilung, Notizen, Dozent seit.
/F120/ Wenn ein Kunde oder eine Firma eine Seminarveranstaltung bucht, dann *muss* das System folgende Buchungsdaten (maximal 500.000) permanent speichern: Angemeldet am, Bestätigung am, Rechnung am, Abgemeldet am, Mitteilung am.
/F130/ Das System *muss* fähig sein, dem Buchhaltungssystem Rechnungsdatensätze mindestens einmal am Tag zur Verfügung zu stellen.

/F140/ Wenn ein Dozent eine Seminarveranstaltung leitet, dann *muss* das System dies speichern.
/F150/ Wenn ein Kunde oder eine Firma im Zahlungsverzug ist, dann *muss* das System folgende Daten dazu speichern: Datum der Rechnung, die noch nicht bezahlt ist, sowie Betrag der Rechnung.
/F160/ Minimal 3 Tage vor einer Veranstaltung *muss* das System dem Kundensachbearbeiter und dem betreffenden Dozenten die Möglichkeit bieten, eine Teilnehmerliste für die Veranstaltung mit folgenden Daten zu erstellen: Seminartitel, Datum von, Datum bis, Veranstaltungsort, Dozent(en) Pro Teilnehmer: Name, Vorname, Firma, Ort.
/F170/ Nach dem Ende einer Veranstaltung *muss* das System dem Kundensachbearbeiter und dem betreffenden Dozenten die Möglichkeit bieten, eine Teilnehmerurkunde für jeden Veranstaltungsteilnehmer mit folgenden Daten zu erstellen: Anrede, Titel, Vorname, Nachname, von Datum, bis Datum, Seminartitel, Veranstaltungsort, Inhaltsübersicht, Veranstaltungsleiter.
/F180/ Das System *muss* dem Dozenten und dem Veranstaltungsbetreuer die Möglichkeit bieten, für eine Veranstaltung Beurteilungsbögen auszudrucken.
/F190/ Wurde eine Veranstaltung durchgeführt, dann *muss* das System fähig sein, eine Honorarmitteilung an die Buchhaltung zu senden.

5 Qualitätsanforderungen
Die Qualitätsanforderungen sind in der Tab. 7.0-2 aufgeführt.
/QF10/ Beim Zugriff über das Internet muss das System eine sichere Übertragung (z. B. https) ermöglichen.
/QF20/ Das System muss die Rollen entsprechend der Tab. 7.0-3 unterscheiden und die dazugehörigen Zugriffsrechte sicherstellen können.
/QF30/ Wenn ein Benutzer SemOrg nutzen will, dann muss das System eine Autorisierung vom Benutzer verlangen.
/QB10/ Die Grundsätze der DIN EN ISO 9241–110 von 2006 mit dem Titel »Ergonomie der Mensch-System-Interaktion – Teil 110: Grundsätze der Dialoggestaltung« sind einzuhalten.
/QE10/ (/LQE10/) Alle Reaktionszeiten auf Benutzeraktionen müssen unter 5 Sekunden liegen.

6 Abnahmekriterien
/A10/ Gültiges Abnahmeszenario: Ein Seminar neu erfassen, eine Veranstaltung neu erfassen, die Veranstaltung dem Seminar zuordnen, einen Dozenten erfassen und dem Seminar und der Veranstaltung zuordnen.
/A20/ TBD usw.

Tab. 7.0-2: Qualitätsanforderungen an SemOrg.

Systemqualität	sehr gut	gut	normal	nicht relevant
Funktionalität				
Angemessenheit		X		
Genauigkeit			X	
Interoperabilität			X	
Sicherheit		X		
Konformität			X	
Zuverlässigkeit				
Reife		X		
Fehlertoleranz			X	
Wiederherstellbarkeit		X		
Konformität			X	
Benutzbarkeit				
Verständlichkeit	X			
Erlernbarkeit			X	
Bedienbarkeit		X		
Attraktivität		X		
Konformität			X	
Effizienz				
Zeitverhalten	X			
Verbrauchsverhalten			X	
Konformität			X	
Wartbarkeit				
Analysierbarkeit	X			
Änderbarkeit			X	
Stabilität	X			
Testbarkeit			X	
Konformität			X	
Portabilität				
Anpassbarkeit	X			
Installierbarkeit			X	
Koexistenz	X			
Austauschbarkeit			X	
Konformität			X	

Tab. 7.0-3: Rollen und Zugriffsrechte in SemOrg.

Rolle	Rechte
Kundensachbearbeiter	/F11/, /F12/, /F20/, /F30/, /F61/, /F70/, /F80/, /F90/, /F100/, /F120/, /F130/, /F150/, /F160/, /F170/
Seminarsachbearbeiter	/F40/, /F50/, /F60/, /F61/, /F90/, /F100/, /F110/, /F140/, /F170/
Dozenten	/F110/, /F140/, /F160/, /F170/, /F180/
Veranstaltungsbetreuer	/F160/, /F170/, /F180/, /F190/
Interessent, Kunde	/F10/

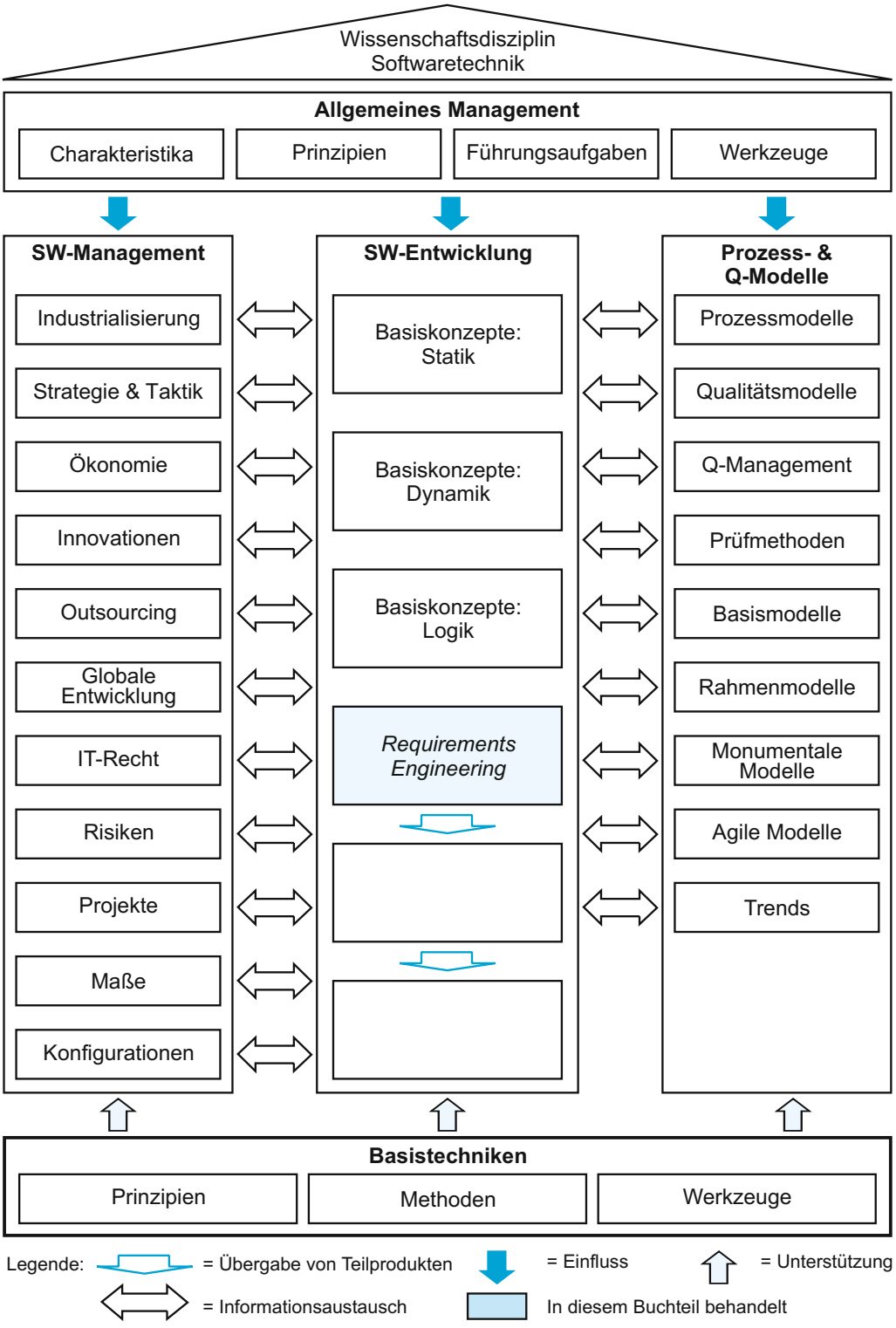
7 Subsystemstruktur (optional)

/K10/ Das System kann in 3 Stufen erstellt werden. Die Funktionalität muss entsprechend der Tab. 7.0-4 realisiert werden. In der Stufe 1 wird die Kernfunktionalität erstellt, in der Stufe 2 wünschenswerte Erweiterungen für den Veranstaltungsbetreuer. In Stufe 3 wird das System für Interessenten und Kunden im Internet geöffnet.

Funktionalität	SemOrg V1.0 (Kern)	SemOrg V2.0	SemOrg V3.0
/F10/ Internet			X
/F11/	X		
/F12/	X		
/F20/	X		
/F30/	X		
/F40/	X		
/F50/	X		
/F60/ Hotels & Räume		X	
/F61/	X		
/F70/	X		
/F80/	X		
/F90/	X		
/F100/	X		
/F110/	X		
/F120/	X		
/F130/	X		
/F140/	X		
/F150/	X		
/F160/ Teilnehmerliste		X	
/F170/ Teilnehmerurkunde		X	
/F180/ Beurteilungsbögen		X	
/F190/	X		

Tab. 7.0-4: Funktionalität pro Version.

IV Requirements Engineering



Terminologie

Die Anforderungen an ein neues Softwareprodukt zu ermitteln, zu spezifizieren, zu analysieren, zu validieren und daraus eine fachliche Lösung abzuleiten bzw. ein Produktmodell zu entwickeln, gehört mit zu den anspruchsvollsten Aufgaben innerhalb der Softwaretechnik. Für diese Tätigkeiten gibt es unterschiedliche Begriffe. Lange Zeit wurde in Deutschland dafür hauptsächlich der Begriff »Systemanalyse« verwendet. Der Begriff ist aber etwas missverständlich, da der Systembegriff oft mehr umfasst als ein Softwaresystem. In der deutschsprachigen Literatur wird heute fast ausschließlich der englische Begriff ***Requirements Engineering*** verwendet (abgekürzt **RE**) – den ich deswegen ebenfalls verwende. Ergänzend oder untergeordnet wird oft noch der Begriff *Requirements Management* benutzt, der auf die Managementaktivitäten im Rahmen des *Requirements Engineering* fokussiert. Die Managementaktivitäten werden *nicht* in diesem Buch, sondern in dem Buch »Lehrbuch der Softwaretechnik – Softwaremanagement« behandelt.

Bevor auf das *Requirements Engineering* eingegangen wird, wird der Sichtwechsel zwischen Problem und Lösung im Rahmen einer Softwareentwicklung aufgezeigt:

■ »Problem vs. Lösung«, S. 437

Die Bedeutung des *Requirements Engineering* erkennt man daran, wie stark der Erfolg einer Softwareentwicklung davon abhängt:

■ »Bedeutung, Probleme und Best Practices«, S. 439

Im Rahmen des *Requirements Engineering* sind eine Reihe von Aktivitäten durchzuführen, die zu Ergebnissen führen:

■ »Aktivitäten und Artefakte«, S. 443

Alle diese Aktivitäten sind eingebettet in einen Prozess, der je nach Produktart, Produktumfang, Firmenorganisation und Firmengröße sowie Mitarbeiterqualifikation sehr unterschiedlich sein kann:

■ »Der Requirements Engineering-Prozess«, S. 449

Bevor eine Vorgehensweise im *Requirements Engineering* festgelegt werden kann, muss geklärt werden, was Anforderungen überhaupt sind, wer die Anforderungen festlegt und welche Eigenschaften für ein zu entwickelndes Softwareprodukt zu spezifizieren sind:

■ »Anforderungen und Anforderungsarten«, S. 455

Damit Anforderungen analysiert, überprüft und validiert werden können, müssen sowohl jede Anforderung für sich als auch alle Anforderungen zusammen gewisse Qualitätskriterien erfüllen:

■ »Anforderungen an Anforderungen«, S. 475

Neben den Qualitätskriterien muss jede Anforderung es ermöglichen, bestimmte Attribute zu erfassen:

■ »Anforderungsattribute«, S. 479

In der Praxis werden Anforderungen natürlichsprachlich formuliert. Die damit verbundenen Probleme und Möglichkeiten ihrer Vermeidung sollten bekannt sein:

- »Natürlichsprachliche Anforderungen«, S. 481

Um beim Spezifizieren der Anforderungen wichtige Gesichtspunkte nicht zu vergessen und um immer gleichartig aufgebaute Produktspezifikationen zu haben, gibt es geeignete Schablonen:

- »Anforderungsschablonen«, S. 485

Eine Vorgehensweise, um systematisch von der Anforderungsermittlung bis zum fertigen Produktmodell zu gelangen, hängt von vielen Randbedingungen ab:

- Liegt eine Ausschreibung vor, dann gibt es in der Regel bereits ein vom Auftraggeber erstelltes Lastenheft.
- Beauftragt eine Fachabteilung die interne IT-Abteilung mit der Softwareerstellung, dann gibt es außer Ideen der Fachabteilung vielleicht noch keine strukturierte schriftliche Unterlage.
- Gibt es bereits ein eingesetztes Softwaresystem, das abgelöst oder verbessert werden soll?
- Ist eine Individualsoftware zu entwickeln oder sind kundenspezifische Anpassungen einer Standardsoftware vorzunehmen?
- Handelt es sich um eine innovative Softwareentwicklung, für die es keine Vorbilder gibt?

Diese Randbedingungen beeinflussen die Methodik. Es kann daher *keine* allgemeingültige Vorgehensweise geben. Im Folgenden wird ein Vorgehensrahmen vorgestellt, der an die jeweiligen Rahmenbedingungen geeignet angepasst werden muss.

Das Ermitteln und Spezifizieren der Anforderungen gliedert sich in mehrere Aktivitäten:

- »Anforderungen ermitteln und spezifizieren«, S. 503

Anschließend müssen die spezifizierten Anforderungen überprüft werden:

- »Anforderungen analysieren, validieren und abnehmen«, S. 513

Um ein *Go* oder ein *Stop* für eine Produktentwicklung geben zu können, ist es nötig, den voraussichtlichen Aufwand zu schätzen:

- »Schätzen des Aufwands«, S. 515

Außerdem ist es nötig, die Anforderungen mit Prioritäten zu versehen:

- »Anforderungen priorisieren«, S. 543

Sind alle Anforderungen spezifiziert, analysiert, validiert und abgenommen, dann kann auf dieser Grundlage eine fachliche Lösung erarbeitet werden. Dies geschieht in der Regel durch eine Modellierung:

- »Anforderungen modellieren«, S. 547

Die fachliche Lösung der beiden Fallstudien wird präsentiert:

- »Fallstudie: SemOrg V1.0 – Die fachliche Lösung«, S. 565
- »Fallstudie: Fensterheber – Die fachliche Lösung«, S. 575

Anschließend müssen die modellierten Anforderungen einer Qualitätsüberprüfung unterzogen werden:

- »Modellierte Anforderungen analysieren, verifizieren und abnehmen«, S. 587

Nach der Abnahme der fachlichen Lösung sind die Aufgaben des *Requirements Engineering* beendet.

Ausbildung

Ein einheitliches Berufsbild für einen *Requirements Engineer* gibt es noch nicht [Paec08]. Ein »International Requirements Engineering Board (IREB) e.V.« hat Vorschläge für ein Curriculum veröffentlicht. Der »Lehrplan IREB Certified Professional for Requirements Engineering – Foundation Level« definiert ein Basiswissen und kann durch ein Zertifikat bescheinigt werden, siehe IREB-Website (`http://certified-re.de/`). Die Inhalte dieses Lehrbuchs sowie des »Lehrbuchs der Softwaretechnik – Softwaremanagement« decken weite Teile dieses Lehrplans ab.

12 Problem vs. Lösung

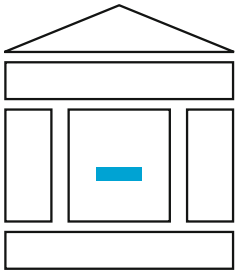
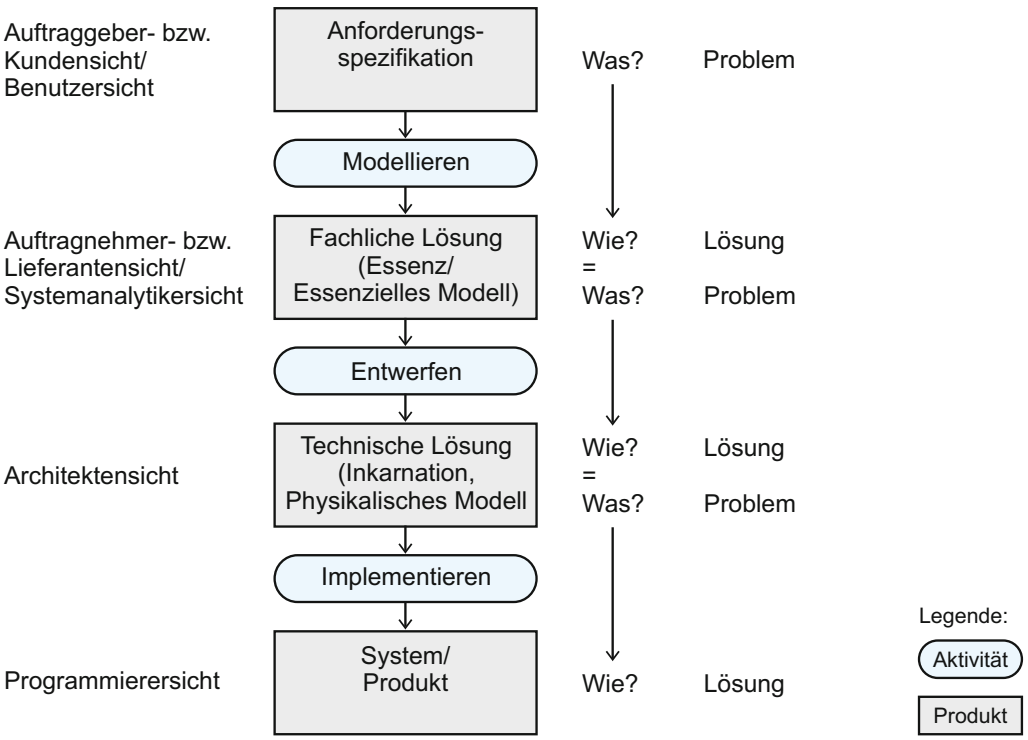


Abb. 12.0-1: Was vs. Wie im Entwicklungsprozess.

Was vs. Wie oder Problem vs. Lösung

Anforderungen legen fest, welche Eigenschaften ein zu entwickelndes Softwaresystem besitzen soll. Sie beschreiben also das Problem, das gelöst werden soll. Die Lösung des Problems, d. h. das »Wie« ist die fachliche Lösung. Die fachliche Lösung wiederum stellt das Problem für die technische Lösung dar usw. Die Abb. 12.0-1 veranschaulicht die verschiedenen Sichtweisen.



Wie die Abb. 12.0-1 zeigt, alterniert der Entwicklungsprozess zwischen Problem und Lösung. Die Lösung auf einem höheren Abstraktionsniveau ist das Problem für das jeweils niedrigere Abstraktionsniveau.

Lösungsraum

Im Laufe der Softwareentwicklung wird der Lösungsraum immer mehr eingeschränkt. Daher ist es wichtig, dass bei der Aufstellung der Anforderungen möglichst *keine* Einschränkungen für die nachfolgenden Aktivitäten erfolgen. Die Abb. 12.0-2 zeigt, wie der Lösungsraum zunehmend verringert wird.

H. Balzert, *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*,
DOI 10.1007/978-3-8274-2247-7_12, © Spektrum Akademischer Verlag Heidelberg 2009

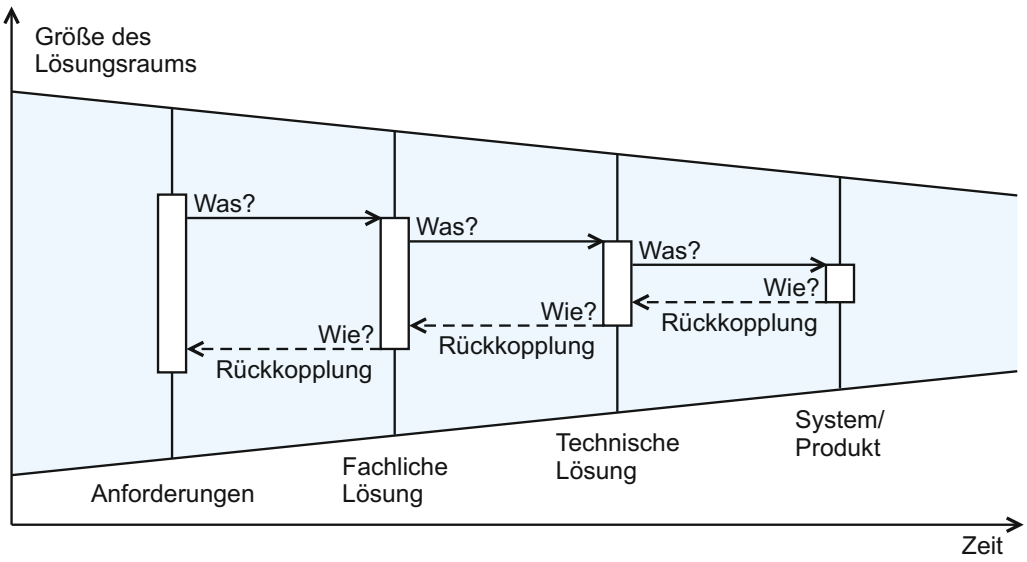


Abb. 12.0-2: Einschränkung des Lösungsraums während der Entwicklung.

Wechselwirkungen

Zwischen dem jeweiligen Problemraum und dem Lösungsraum gibt es Abhängigkeiten. Es kann durchaus sein, dass beim Versuch, für ein Problem eine Lösung zu finden, es sich herausstellt, dass das Problem so nicht gelöst werden kann. Es muss dann das Problem modifiziert werden oder die Softwareentwicklung muss eingestellt werden. Diese Abhängigkeiten zeigt die Abb. 12.0-3.

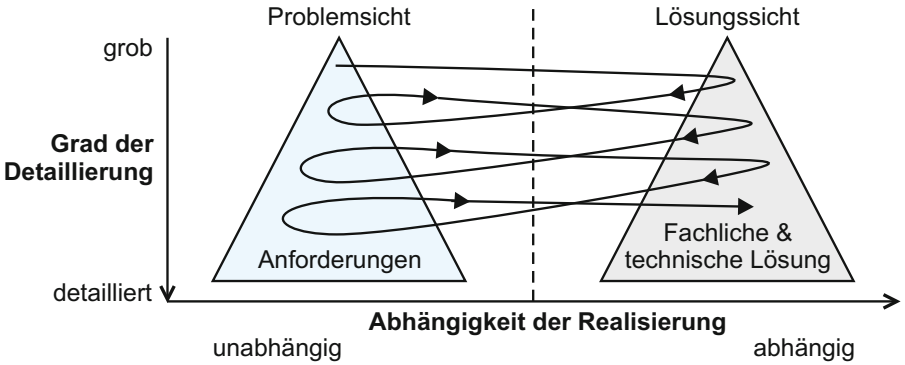
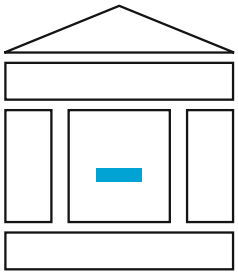


Abb. 12.0-3: Wechselwirkung zwischen Anforderungen und fachlicher und technischer Lösung (in Anlehnung an [Nuse01]).

Technik vs. Management

Bei einer Softwareentwicklung sollte klar zwischen den technischen Aspekten und den Managementaspekten unterschieden werden. Anforderungen und Festlegungen, die den Entwicklungsprozess betreffen, gehören in eigene Dokumente. Beispiele für solche Anforderungen sind: Kosten, Fertigstellungstermine, Berichtsverfahren, Festlegung von Entwicklungs- und Qualitätssicherungsmethoden, Kriterien für die Validation und Verifikation, Abnahmeverfahren [IEEE830, S. 10].

13 Bedeutung, Probleme und *Best Practices*



Empirie

Es gibt eine Vielzahl von Untersuchungen darüber, wie erfolgreich oder nicht erfolgreich Softwareprojekte waren und sind. Eine berühmte Untersuchung, die immer wieder zitiert wird, ist der CHAOS-Report von 1994 [Stan94]. In diesem Report wird gesagt, dass 31 Prozent aller Softwareprojekte *vor* der Fertigstellung abgebrochen werden. Dieser Report wurde wegen fehlender Transparenz, nicht offen gelegter Methodik und anderer Mängel kritisiert. Eine neue Untersuchung kommt zu folgenden Erkenntnissen [EmKo08]:

- Es wurden in den Jahren 2005 und 2007 zwei internationale Befragungen durchgeführt.
- Die Hälfte aller Projekte dauerte neun Monate oder weniger.
- Die Anzahl der Entwickler lag zwischen drei und 10, wobei die meisten Projekte weniger als 10 Entwickler hatten.
- Von allen Projekten wurden 2005 **16 Prozent** und 2007 **12 Prozent** abgebrochen, *bevor* irgendetwas ausgeliefert wurde.
- Es wurde *keine* statistisch signifikante Differenz der Abbruchwahrscheinlichkeit bezogen auf die Projektdauer und die Anzahl der Mitarbeiter festgestellt.
- Die vier Hauptgründe für einen Projektabbruch waren:
  - **Änderungen der Anforderungen** und des Umfangs (33 Prozent).
  - Mangelnde Einbindung des höheren Managements (33 Prozent).
  - Engpass im Budget (28 Prozent).
  - Fehlende Projektmanagement-Fähigkeiten (28 Prozent).
- Zwischen 48 Prozent (2005) und 55 Prozent (2007) der ausgelieferten Projekte waren erfolgreich, während zwischen 17 und 22 Prozent der ausgelieferten Projekte *nicht* erfolgreich waren.
- Kombiniert man die abgebrochenen Projekte mit den nicht erfolgreichen Projekten, dann ergeben sich für 2005 **34 Prozent** und für 2007 **26 Prozent**. Dies ist eine hohe Misserfolgsrate für eine angewandte Disziplin.
- Seit 1994 gibt es einen klaren Trend hin zu abnehmenden Abbruchraten – belegbar über alle Studien hinweg.

Zitat

»Stolze 43 Prozent der im Betrieb festgestellten Fehler in Steuerungs- und Regelungssoftware sind auf Unzulänglichkeiten in der Analysephase oder der Systemspezifikation zurückzuführen. 'Sie wirken sich viel später aus und kosten eine Menge Geld, wenn man sie beheben will'« [cz06].

H. Balzert, *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*,
DOI 10.1007/978-3-8274-2247-7_13, © Spektrum Akademischer Verlag Heidelberg 2009

Eine Befragung von 80 Unternehmen im Jahr 2008 durch die Fachhochschule St. Gallen führte zu folgenden Ergebnissen ([Ott08], [Huth09]):

Fehlerquellen

- Die häufigste Fehlerquelle bei der Anforderungserfassung sind mit 83 Prozent **sprachliche Fehler** (Unverständlichkeit, Missverständlichkeit, Unquantifizierbarkeit).
- **Logische Fehler** (Widersprüchlichkeit, Redundanz) bei der Anforderungserfassung treten mit 75 Prozent auf.
- **Inhaltliche Fehler** (Falsche Sachverhalte, Unvollständigkeit) machen 73 Prozent aus.

Änderungen

- Änderungen der Anforderungen sind die Regel.
- 77 Prozent der Änderungen werden durch Missverständnisse in der Kommunikation der ***Stakeholder*** verursacht (siehe auch »Anforderungen ermitteln und spezifizieren«, S. 503).
- Wachsende oder sich ändernde Anforderungen an das Gesamtsystem (Produkt ändert sich, Zielgruppe ändert sich usw.) führen in 70 Prozent aller Fälle zu geänderten Anforderungen.
- Pilotbetrieb, Prototypen, Analysen, zusätzliche Know-how-Träger usw. führen zu neuen Erkenntnissen, die in 66 Prozent der Fälle zu Anforderungsänderungen führen.
- 56 Prozent der Unternehmen gaben an, dass eine Änderung im Budget, der Priorisierung oder der Marketingstrategie immer oder oft die Ursache für eine geänderte Anforderung ist.
- In 50 Prozent aller Fälle müssen Anforderungen wegen ungenauen Formulierungen oder falschen Einschätzungen bzgl. der Machbarkeit geändert werden.

Nutzen von RE

Zwei Studien der NASA von 1997 und 2001 haben Folgendes ergeben [Eber08, S. 9 f.]:

- Wenn die NASA in ihre Projekte weniger als 5 Prozent Vorbereitungsaufwand – insbesondere für die Anforderungen – investiert, dann führen die Projekte zu starken Verzögerungen. Liegt der Vorbereitungsaufwand zwischen 10 und 20 Prozent des gesamten Projektaufwands, dann liegen die Terminverzüge unter 30 Prozent.
- Projekte mit 5 Prozent Aufwand für RE führen zu Kostenüberschreitungen zwischen 80 und 200 Prozent. Liegt der RE-Aufwand zwischen 8 und 14 Prozent, dann liegt die Kostenüberschreitung unter 60 Prozent.

Aufwand von RE

Eine Feldstudie mit 15 RE-Teams, davon sechs Teams, die Standardsoftware entwickelt haben, und neun Teams, die Individualsoftware entwickelt haben, führte zu folgenden Erkenntnissen [HoLe01]:

- 1981 wurden 6 Prozent der Projektkosten und 9 bis 12 Prozent der Projektdauer für das *Requirements Engineering* verwendet.

- 2001 wurden **16 Prozent der Projektkosten** und **39 Prozent der Projektdauer** für RE aufgewendet. In 20 Jahren sind die Ressourcen für RE also signifikant gestiegen.
- Die Größe des RE-Teams liegt im Durchschnitt bei 5 bis 6 Personen, während die gesamte Projektteamgröße bei ca. 17 Personen liegt.

7 RE-Risiken

In [Eber08, S. 4 ff.] und [LWE01] werden sieben Risiken aufgeführt, die im RE vermieden werden sollten:

- Risiko 1: Kunden sind im Projekt unzureichend repräsentiert.
- Risiko 2: Kritische Anforderungen werden übersehen.
- Risiko 3: Es werden nur funktionale Anforderungen berücksichtigt.
- Risiko 4: Anforderungen werden unkontrolliert geändert.
- Risiko 5: Anforderungen beschreiben den Entwurf.
- Risiko 6: Anforderungen werden nicht auf Qualität geprüft.
- Risiko 7: Anforderungen werden perfektioniert.

Probleme

Neben diesen Risiken gibt es ein generelles interkulturelles bzw. kommunikatives Problem bei der Ermittlung der Anforderungen (siehe z. B. [Dahm00]). In der Regel ist die Kommunikation zwischen dem Auftraggeber bzw. der Fachabteilung und dem Auftragnehmer, d. h. den Softwareentwicklern, problembehaftet. Folgende Probleme treten oft auf:

- Der Auftraggeber weiß zu Projektbeginn *nicht* genau, was er will. Oft hört man die Aussage des Auftraggebers: »Wenn ich die Software sehe, dann kann ich sagen, was ich will.«
- Der Auftraggeber kann das, wovon er weiß, dass er es will, *nicht* vollständig mitteilen.
- Der Auftraggeber versteht *nicht*, was der Softwareentwickler außer den vorgelegten Beispielen noch leisten könnte.
- Der Auftraggeber weiß *nicht*, welche Software möglich wäre, wenn der Softwareentwickler besser über seine Bedürfnisse informiert wäre.

Zitat

»Eine recht naive und gefährliche Annahme versteckt sich außerdem in der Zuversicht, daß Entwickler und Anwender es schon rechtzeitig merken werden, wenn Mißverständnisse auftreten, und diese durch Nachfragen leicht beseitigen können. Vielmehr ist es die Regel, daß Verständnislücken auf beiden Seiten viel zu spät und manchmal gar nicht aufgeklärt werden, so daß für den Anwender Unnützes mit viel Mühe implementiert wird, und zugleich zentrale, aber bislang implizit gebliebene Anforderungen gerade noch nicht erfüllt sind. [...] Um brauchbare Software herstellen zu können, sollte die Kultur der Anwender berücksichtigt werden, da die Software für den Anwender entwickelt wird und in seine Kultur eingebettet werden soll. Dazu müssen sich aber beide dieser Situation bewußt sein.« [Dahm00, S. 175].

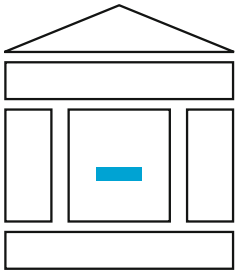
Best Practices

Nach [HoLe01, S. 65] wenden die besten RE-Teams die in der Tab. 13.0-1 aufgeführten ***Best Practices*** an.

Fokus	*Best Practice*	Kosten der Einführung	Kosten der Anwendung	Hauptnutzen
Wissen	Kunden & Benutzer einbinden	Gering	Mittel	Besseres Verständnis der »echten Bedürfnisse«
Wissen	Identifizieren & Konsultieren aller möglichen RE-Quellen	Gering – Mittel	Mittel	Verbesserte Überdeckung der Anforderungen
Wissen	Erfahrene Projektmanager & Teammitglieder einsetzen	Mittel – Hoch	Mittel	Besser voraussagbare Leistung
Ressourcen	15–30 % der gesamten Ressourcen für RE	Gering	Mittel – Hoch	Beibehaltung einer qualitativ hochwertigen Spezifikation während des Projekts
Ressourcen	Anforderungsschablonen & Beispiele zur Verfügung stellen	Gering – Mittel	Gering	Verbesserte Spezifikationsqualität
Ressourcen	Gute Beziehungen zu allen Beteiligten & Betroffenen herstellen	Gering	Gering	Bessere Zufriedenstellung der Kundenbedürfnisse
Prozess	Anforderungen priorisieren	Gering	Gering – Mittel	Aufmerksamkeit auf die wichtigsten Kundenwünsche fokussieren
Prozess	Ergänzende Modelle zusammen mit Prototypen entwickeln	Gering – Mittel	Mittel	Eliminieren von Spezifikationswidersprüchen & Inkonsistenzen
Prozess	Eine Nachverfolgungsmatrix pflegen	Mittel	Mittel	Explizite Verweise zwischen Anforderungen & Ergebnissen
Prozess	*Peer Reviews* durch Benutzer, Szenarien & *Walk-Throughs* zum Validieren & Verifizieren der Anforderungen	Gering	Mittel	Genauere Spezifikationen & höhere Kundenzufriedenheit

Tab. 13.0-1: Best Practices im RE.

14 Aktivitäten und Artefakte



Im Rahmen einer Softwareentwicklung müssen Aktivitäten durchgeführt werden, die zu Ergebnissen – im Folgenden **Artefakte** *(artifacts)* genannt – führen. Eine Aktivität wird durch Mitarbeiter ausgeführt, die definierte **Rollen** einnehmen. Unter Beachtung von Methoden, Richtlinien, Konventionen, Checklisten und Schablonen wird – ausgehend von einem oder mehreren gegebenen Artefakten oder Informationen – ein neues Artefakt erstellt oder der Zustand oder der Inhalt gegebener Artefakte geändert. Für die Durchführung der Aktivität werden in der Regel vom Softwaremanagement vorgegebene Werkzeuge eingesetzt (Abb. 14.0-1).

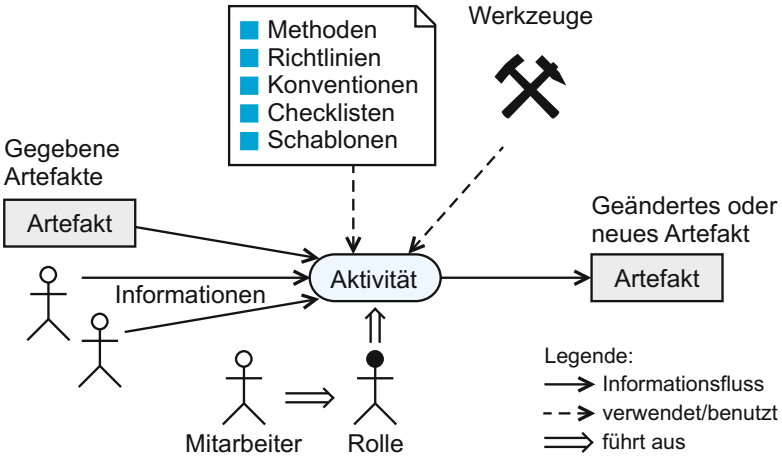


Abb. 14.0-1: Durchführung einer Aktivität.

Eine Artefaktbeschreibung legt die Inhalte und das Layout des Artefakts fest. Sie erfolgt nach einer festen Schablone, der **Artefakt-Schablone** *(artifact template)*. Jedes Artefakt enthält einen Artefakt-Vorspann, der die Historie des Artefakts aufzeigt.

Rollen im RE

Die Aufgaben, die im *Requirements Engineering* zu erledigen sind, werden von der **Rolle** des ***Requirements Engineer*** erledigt, im Deutschen auch Systemanalytiker, Anforderungsingenieur oder Anforderungsanalytiker genannt.

Welche Kompetenzen sollte Ihrer Meinung nach ein *Requirements Engineer* besitzen? Frage

Ein *Requirements Engineer* sollte über folgende Kompetenzen verfügen [IREB07, S. 9]: Antwort

- Analytisches Denken
- Selbstbewusstes Auftreten

H. Balzert, *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*,
DOI 10.1007/978-3-8274-2247-7_14, © Spektrum Akademischer Verlag Heidelberg 2009

- **Empathische** Fähigkeiten
- Moderationsfähigkeit
- Überzeugungsfähigkeit
- Kommunikationsfähigkeiten
- Sprachliche Kompetenz
- Methodische Kompetenz

In [Balz08] werden folgende Kompetenzen genannt:

- Abstraktes Denken (Vom Konkreten zum Abstrakten)
- Flexibilität
- Hohe Kommunikationsbereitschaft und -fähigkeit
- Hineindenken in andere Begriffs- und Vorstellungswelten
- Fachwissen aus den Anwendungsgebieten

Ein *Requirements Engineer* wird oft durch einen oder mehrere **Anwendungsspezialisten** unterstützt, die spezielles Wissen der Domäne besitzen, für die das Produkt entwickelt werden soll. Ein Anwendungsspezialist sollte über folgende Kompetenzen verfügen:

- Abstraktes Denken (Vom Konkreten zum Abstrakten)
- Ganzheitliches Denken, z. B. in Geschäftsprozessen
- Hohe Kommunikationsbereitschaft
- Breites Modellierungswissen über das Anwendungsgebiet
- Kenntnis vorhandener Softwaresysteme für das Anwendungsgebiet
- Fähigkeit, das Anwendungsgebiet aufgabengerecht zu strukturieren
- Automatisierungsmöglichkeiten des Anwendungsgebiets einschätzen können unter Berücksichtigung von wirtschaftlichen und ergonomischen Gesichtspunkten

Die Rollen *Requirements Engineer* und Anwendungsspezialist werden oft nicht unterschieden, sondern zu einer Rolle zusammengefasst.

Aktivitäten im RE

Unabhängig von einem Prozessmodell sind im *Requirements Engineering* folgende Aktivitäten durchzuführen:

- **Anforderungen ermitteln**: Die Anforderungen müssen von den Beteiligten und Betroffenen sowie sonstigen Quellen, z. B. Normen, Gesetzestexte, Standards, systematisch gewonnen werden.
- **Anforderungen spezifizieren**: Die ermittelten Anforderungen müssen spezifiziert werden, d. h. unter Berücksichtigung von festgelegten Methoden, Richtlinien, Konventionen, Checklisten und Schablonen beschrieben werden.
- **Anforderungen analysieren, validieren und abnehmen**: Die spezifizierten Anforderungen müssen anhand von Richtlinien und Checklisten analysiert werden, um Fehler, Missverständnis-

se und Unklarheiten zu beseitigen. Die **Validation** hat das Ziel, die Eignung bzw. den Wert des spezifizierten Produkts auf seinen Einsatzzweck hin zu überprüfen.

- **Anforderungen modellieren**: Die analysierten und validierten Anforderungen bilden den Ausgangspunkt für die Modellierung der fachlichen Lösung.
- **Anforderungen analysieren, verifizieren und abnehmen**: Das entstehende Modell wird permanent analysiert, führt zu Rücksprachen und evtl. Änderungen an der Spezifikation. Außerdem wird das Modell gegen die Spezifikation verifiziert. Der Begriff **Verifikation** bedeutet in diesem Kontext die Überprüfung der Übereinstimmung zwischen einem Softwareprodukt und seiner Spezifikation.
- **Anforderungen verwalten und managen**: Alle Anforderungen müssen verwaltet werden – was in der Regel mit Hilfe einer Softwareentwicklungsumgebung oder eines Softwarewerkzeugs passiert. Unter Managen ist die Änderung, Nachverfolgbarkeit, die Versionierung usw. der Anforderungen zu verstehen.

Alle Aktivitäten zusammen werden oft auch **Spezifizieren** oder **Definieren der Anforderungen** genannt.

Artefakte des RE

In Abhängigkeit vom verwendeten Prozessmodell, von der Art der Auftraggeber-Auftragnehmer-Situation und von der zu entwickelnden Produktart sind die Artefakte sowohl bezogen auf die Anzahl als auch bezogen auf den Aufbau unterschiedlich. Grobgranular lassen sich zwei Artefakte unterscheiden:

- Die **Anforderungsspezifikation** *(requirements specification)* bzw. die **Produktspezifikation**: Enthält alle ermittelten, spezifizierten, analysierten und validierten Anforderungen aus der Kunden- bzw. Auftraggebersicht. In die Anforderungsspezifikation integriert oder als eigenständiges Artefakt ist ein **Glossar** der verwendeten domänenspezifischen Fachbegriffe.
- Die **fachliche Lösung** bzw. das Produktmodell oder das **essenzielle Modell** *(essential model)*: Beschreibt die analysierte und verifizierte fachliche Lösung des Produkts aus der Auftragnehmer- bzw. Lieferantensicht.

Einen grobgranularen Überblick über die Aktivitäten und Artefakte gibt die Abb. 14.0-2.

Hinweis

Oft gibt es keine klare Trennung zwischen Anforderungsspezifikation und fachlicher Lösung. Oder zum *Requirements Engineering* wird nur die Anforderungsspezifikation gezählt. Die fachliche

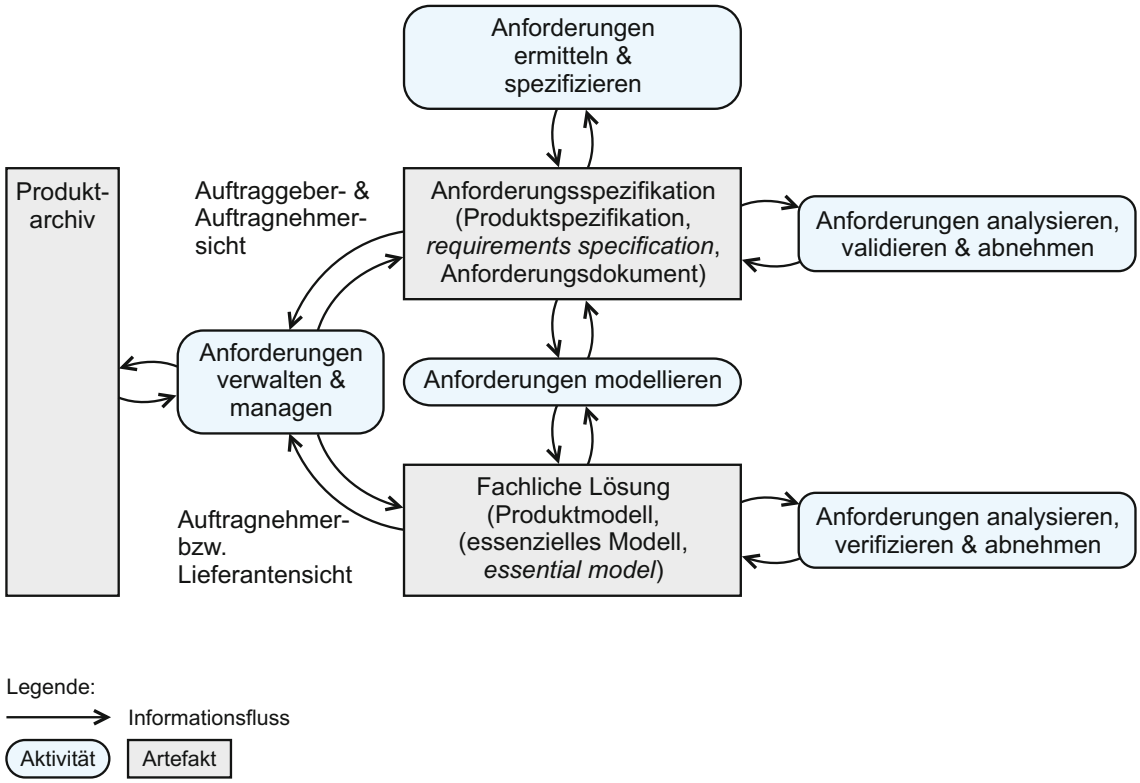


Abb. 14.0-2: Grobgranulare Übersicht über die Aktivitäten und Artefakte des Requirements Engineering.

Lösung und die technische Lösung, d. h. die Produktarchitektur, fließen dann oft ineinander, was aber unbedingt vermieden werden sollte.

Die Anforderungsspezifikation

Bei der Anforderungsspezifikation – oft auch Anforderungsdokument genannt – kann es sich um ein Artefakt, aber auch um zwei Artefakte handeln.

In manchen Prozessmodellen ist das Ermitteln und Spezifizieren der Anforderungen in zwei Teile geteilt (Abb. 14.0-3):

- In eine Ausschreibungs- oder Planungsphase und
- in eine Spezifikationsphase.

Diese Zweiteilung ist immer dann notwendig, wenn ein Auftraggeber eine Softwareentwicklung ausschreibt, d. h. der Auftragnehmer noch nicht feststeht. Der Auftraggeber erstellt dann selbstständig – oder mit Hilfe von Beratern – ein Lastenheft.

Definitionen

Lastenheft: »Zusammenstellung aller Anforderungen des Auftraggebers hinsichtlich Liefer- und Leistungsumfang. Im Lastenheft sind die Anforderungen aus Anwendersicht einschließlich aller Randbedingungen zu beschreiben. Diese sollen quantifizierbar und prüfbar sein. Im Lastenheft wird definiert WAS und WO-

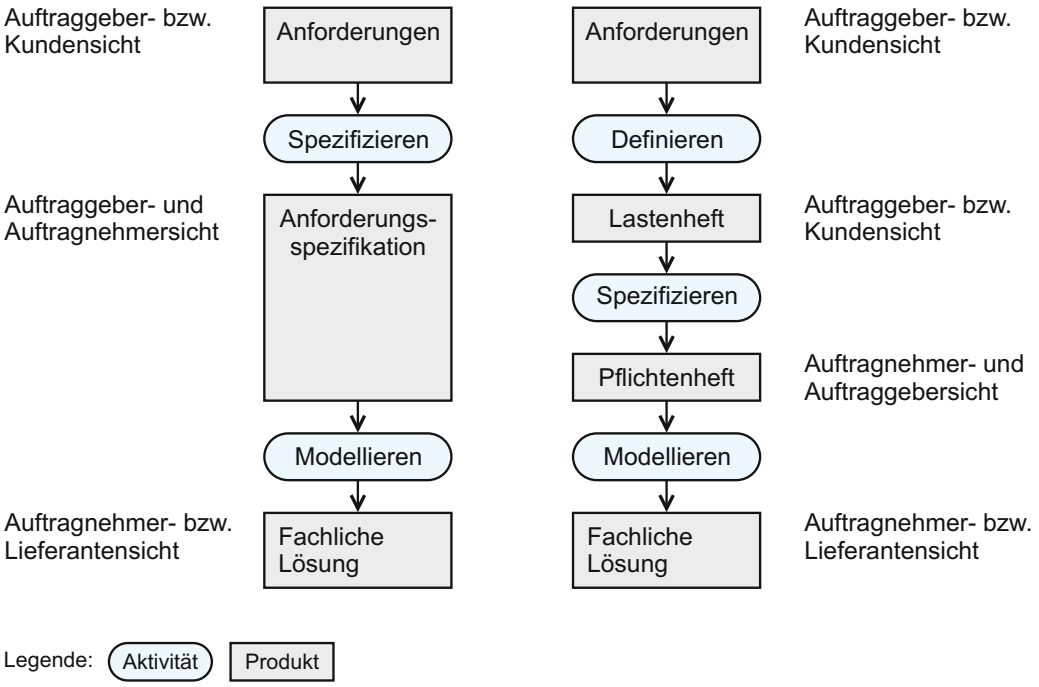


Abb. 14.0-3: Einstufiges (links) und zweistufiges (rechts) Vorgehen im Requirements Engineering.

FÜR zu lösen ist. Das Lastenheft wird vom Auftraggeber oder in dessen Auftrag erstellt. Es dient als Ausschreibungs-, Angebots- und/oder Vertragsgrundlage« [VDI2519, S. 2].

Lastenheft: »Vom Auftraggeber festgelegte Gesamtheit der Forderungen an die Lieferungen und Leistungen eines Auftragnehmers innerhalb eines Auftrags« [DIN69905, S. 3].

Ist der Auftrag erteilt, dann erstellt der Auftragnehmer zusammen mit dem Auftraggeber ein Pflichtenheft, das wesentlich ausführlicher als das Lastenheft ist.

Definitionen

Pflichtenheft: »Beschreibung der Realisierung aller Anforderungen des Lastenheftes. Das Pflichtenheft enthält das Lastenheft. Im Pflichtenheft werden die Anwendervorgaben detailliert und die Realisierungsanforderungen beschrieben. Im Pflichtenheft wird definiert, WIE und WOMIT die Anforderungen zu realisieren sind. [...] Das Pflichtenheft wird in der Regel nach Auftragserteilung vom Auftragnehmer erstellt, falls erforderlich unter Mitwirkung des Auftraggebers. Der Auftragnehmer prüft bei der Erstellung des Pflichtenhefts die Widerspruchsfreiheit und Realisierbarkeit der im Lastenheft genannten Anforderungen. Das Pflichtenheft bedarf der Genehmigung durch den Auftraggeber« [VDI2519, S. 2 f.].

Pflichtenheft: »Vom Auftragnehmer erarbeitete Realisierungsvorgaben aufgrund der Umsetzung des vom Auftraggeber vorgegebenen Lastenheftes« [DIN69905, S. 3].

Liegt eine Trennung Auftraggeber – Auftragnehmer vor, dann ist in der Regel eine zweistufige Spezifikation notwendig. Aber auch wenn diese Situation nicht vorliegt, kann eine zweistufige Spezifikation sinnvoll sein. Bevor eine Softwareentwicklung genehmigt wird – z. B. innerbetrieblich – werden die Anforderungen oft zunächst grob zusammengestellt und danach eine Wirtschaftlichkeitsbetrachtung angestellt. Am Ende einer solchen Planungsphase steht dann das »Stop or Go« für die eigentliche Entwicklung. Für eine solche Planungsphase ist der Aufwand zur Erstellung eines Pflichtenhefts zu hoch. Daher wird zunächst nur ein Lastenheft erstellt.

+ Vorteilhaft bei diesem Vorgehen ist, dass es ein klar abgegrenztes Ende der RE-Phase gibt, zu dessen Zeitpunkt auch der Auftraggeber sich klar für die fachliche Lösung entscheiden muss.
− Nachteilig ist, dass spätere Änderungen immer über den Änderungsprozess abgewickelt werden müssen.

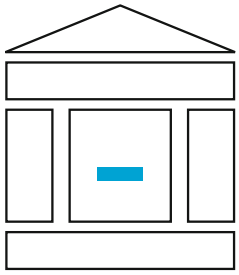
Wichtig ist, dass definierte Anforderungen und fachliche Lösungen in einem Wiederverwendungsarchiv gesammelt werden, um sie projektübergreifend nutzen zu können.

Die fachliche Lösung

Die fachliche Lösung bzw. das Produktmodell besteht – in Abhängigkeit von der verwendeten Methode – in der Regel aus verschiedenen Artefakten:

- OOA-Modell (siehe »OOA-Methode«, S. 559)
- GUI-Konzept oder -Prototyp
- evtl. Benutzerhandbuch

15 Der *Requirements Engineering*-Prozess



In großen Unternehmen und Softwarehäusern ist die Softwareentwicklung eingebettet in die Unternehmens- und Produktstrategie. Der Weg von einer Idee hin zu einem Produkt besteht aus vielen Schritten, in denen Annahmen gemacht, geprüft und substanziiert werden. Aus vielen ursprünglichen Ideen und Visionen werden am Ende nur einzelne wenige Produkte entwickelt. Dies liegt an den stark steigenden Kosten, sobald Entwicklungsprojekte und das begleitende Marketing gestartet werden. Man versucht innerhalb der Strategieentwicklung daher, möglichst viele Ideen frühzeitig zu bewerten und nur für jene mit großem Potenzial die Entwicklung zu starten. Die Abb. 15.0-1 zeigt diese Entwicklung vom Ideenmanagement hin zum konkreten Portfolio von Produkten.

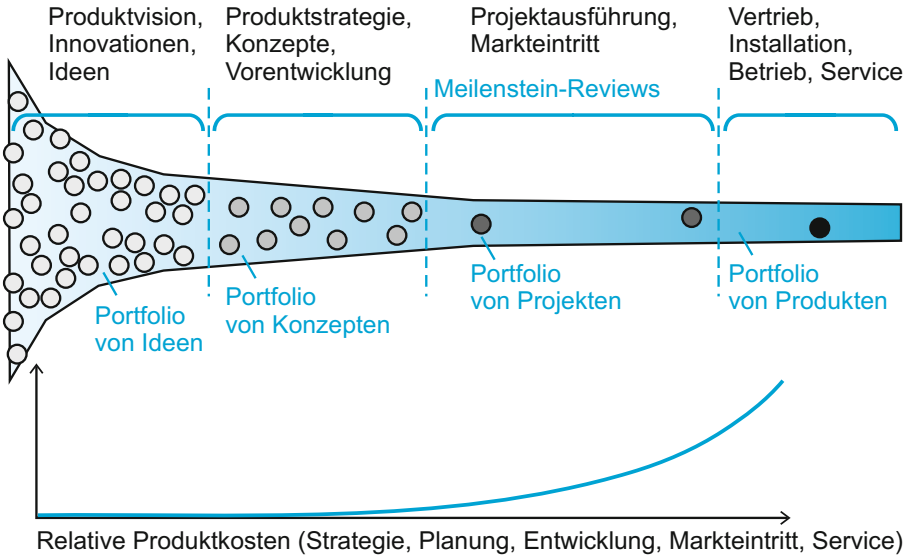


Abb. 15.0-1: Ideen bewerten.

Auf die Aktivitäten im Vorfeld einer Softwareentwicklung wird im »Lehrbuch der Softwaretechnik – Softwaremanagement« eingegangen.

Annahme

Im Folgenden wird davon ausgegangen, dass die prinzipielle Entscheidung für die Entwicklung eines Softwareprodukts gefallen ist und dass es nun darum geht, die Anforderungen zu ermitteln und eine fachliche Lösung zu erstellen. Nach der Ermittlung der Anforderungen und einer Aufwandsschätzung kann u. U. noch eine Überprüfung dahingehend erfolgen, dass ein explizites »Go« oder auch ein »Stop« der weiteren Entwicklung erfolgt.

H. Balzert, *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*,
DOI 10.1007/978-3-8274-2247-7_15, © Spektrum Akademischer Verlag Heidelberg 2009

Granularität

Im Rahmen einer Softwareentwicklung kann der RE-Prozess grob- oder feingranular sein. Traditionell endet der RE-Prozess mit Fertigstellung und Abnahme der **fachlichen Lösung**. Anschließend folgt der Entwurfs-Prozess.

Alle Erfahrungen haben aber gezeigt, dass Anforderungen während der Entwicklung von den *Stakeholdern* noch häufig geändert werden.

Frage

Wie hoch schätzen Sie die Änderungsquote der Anforderungen pro Monat?

Antwort

Anforderungen ändern sich während einer laufenden Softwareentwicklung typischerweise mit 1 bis 5 Prozent des Projektumfangs (Aufwand) pro Monat [Eber08, S. 331].

Daraus ergibt sich, dass entweder der RE-Prozess parallel zur weiteren Entwicklung weiterlaufen muss oder dass es ein definiertes Änderungsmanagement gibt.

Alternativen

Für den RE-Prozess gibt es daher folgende Alternativen (vgl. [Pohl07, S. 30 ff.]):

- RE als eigenständige Entwicklungsphase + definiertes Änderungsmanagement
- RE entwicklungsbegleitend
- RE produktübergreifend

RE als eigenständige Entwicklungsphase und definiertes Änderungsmanagement

In vielen Prozessmodellen wird das RE als eigenständige Phase am Anfang jeder Softwareentwicklung durchgeführt. Die Anforderungen werden für jede Softwareentwicklung unabhängig von anderen Softwareentwicklungen durchgeführt. Muss die RE-Phase abgeschlossen sein, bevor mit der Entwurfsphase begonnen wird, dann liegt ein sequenzielles Prozessmodell vor. Beginnt die Entwurfsphase bereits, bevor die RE-Phase abgeschlossen ist, dann liegt ein nebenläufiges Prozessmodell vor (Abb. 15.0-2).

Unabhängig von diesen beiden Modellen, kann das RE entweder *alle* Anforderungen an ein Produkt enthalten oder *nur* Anforderungen an ein Teilprodukt. Da nach Abschluss des RE-Prozesses in der Regel kontinuierlich Anforderungsänderungen gefordert werden, muss parallel zu den weiteren Entwicklungsphasen ein definiertes Änderungsmanagement *(Change Management)* etabliert sein oder werden, um systematisch mit Änderungen umzugehen. Im Änderungsmanagement werden Änderungswünsche, Fehler und Probleme, die während der Systementwicklung oder -nutzung auftreten, behandelt und gelöst. Die Abb. 15.0-3 zeigt, wie ein Änderungsmanagement aussehen kann.

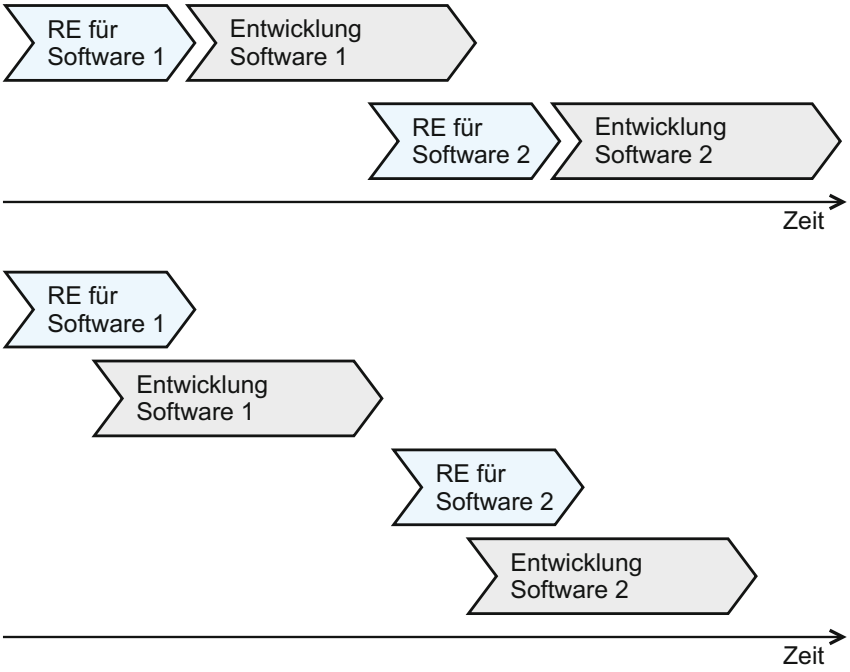


Abb. 15.0-2: Sequenzielles (oben) und nebenläufiges (unten), phasenbezogenes Requirements Engineering (RE).

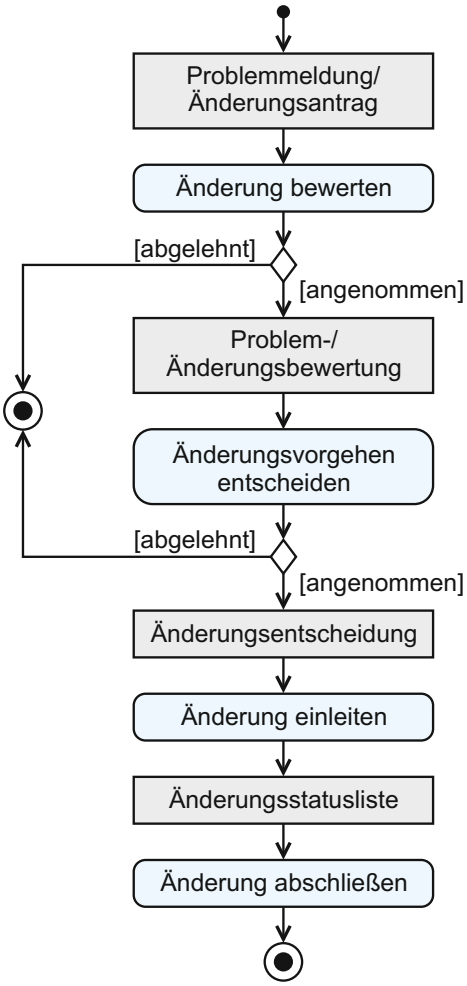


Abb. 15.0-3: Mögliche Zustände einer Änderung und zugehörige Formulare.

RE entwicklungsbegleitend

Beim entwicklungsbegleitenden RE liegt ein phasenübergreifendes *Requirements Engineering* vor, d. h. es handelt sich um eine Querschnittstätigkeit, die den gesamten Entwicklungsprozess begleitet und eine konsistente Erfassung und Verwaltung von Anforderungen sicherstellt (Abb. 15.0-4). Zwischen dem RE und den anderen Entwicklungsphasen gibt es Wechselwirkungen. Änderungen, die sich z. B. aus dem Softwareentwurf ergeben, werden an das RE übergeben.

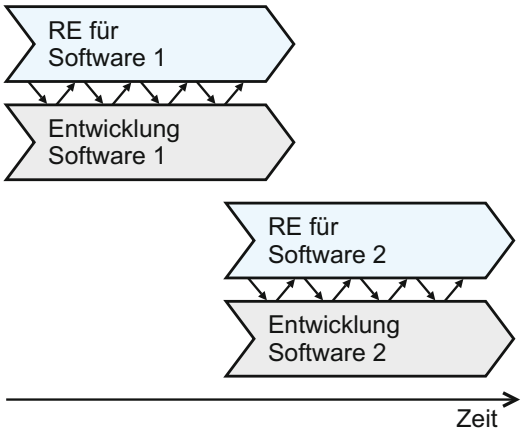


Abb. 15.0-4: Phasenübergreifendes Requirements Engineering.

Der Aufwand für das *Requirements Engineering* fällt also nicht konzentriert am Entwicklungsanfang an, sondern ist über die gesamte Entwicklungszeit verteilt (siehe auch »Schablonen für agile Entwicklungen«, S. 497).

RE produktübergreifend

Befinden sich die zu entwickelnden Softwareprodukte in einer gemeinsamen Domäne, dann ist es sinnvoll, das RE produktübergreifend auszulegen (Abb. 15.0-5). Dies ist in der Regel bei einer Produktlinien-Entwicklung der Fall (siehe auch »Lehrbuch der Softwaretechnik – Softwaremanagement«).

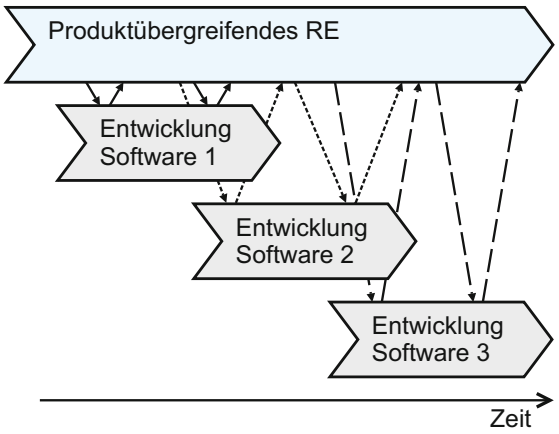


Abb. 15.0-5: Produktübergreifendes Requirements Engineering.

+ Vorteilhaft ist, dass eine fortlaufende Wissensbasis für die Produktdomäne entsteht, auf die bei neuen Produkten sofort zugegriffen werden kann.
+ Die für das RE zuständigen Mitarbeiter können sich ganz auf die Entwicklung und Pflege der RE-Wissensbasis konzentrieren.
− Nachteilig ist, dass ein erhöhter Koordinierungsaufwand zwischen den Projekten entsteht.

16 Anforderungen und Anforderungsarten

Bevor ein Softwaresystem entwickelt werden kann, muss festgelegt werden, welche Anforderungen es erfüllen soll.

Frage

Wie definieren Sie den Begriff »Anforderungen«?

Antwort

Im Duden wird der Begriff wie folgt definiert:

»Anforderung: <meist Plural> das, was man von jmdm. als [Arbeits]leistung erwartet, von ihm verlangt.« (Duden Bedeutungen, PC-Bibliothek).

Übertragen auf ein Softwaresystem lässt sich der Begriff wie folgt definieren:

Definition

Anforderungen *(requirements)* legen fest, was man von einem Softwaresystem als Eigenschaften erwartet.

Der Begriff Leistung wurde bewusst nicht in die Definition übernommen, da Leistung oft einen Bezug zurzeit hat. Dies trifft auf Softwaresysteme aber nur in Teilbereichen zu.

In dieser Definition sind zwei Dinge noch offen:

1. Wer ist »man«?
2. Was sind »Eigenschaften«?

Beides wurde in der Definition bewusst *nicht* genauer festgelegt, da es dazu verschiedene Antworten gibt. Man kann die Worte »man« und »Eigenschaften« auch als Platzhalter verstehen, die je nach Perspektive durch konkretere Begriffe substituiert werden.

Wer ist »man«?

An jeder Softwareentwicklung im industriellen Maßstab wirken verschiedene und meist auch viele Personen mit. Alle Personen und Organisationen, die ein Interesse an einer Softwareentwicklung haben und von einer Softwareentwicklung bzw. dem Einsatz des Softwaresystems betroffen sind, werden mit dem englischen Begriff ***Stakeholder*** bezeichnet. Der Begriff bedeutet im Deutschen etwa: Akteur, Interessenvertreter. »*The term stakeholder generalizes the traditional notion of customer or user in requirements engineering to all parties involved in a system's requirements*« [GlWi07, S. 18].

Gibt es einen individuellen Auftraggeber, dann wird dieser wesentliche Anforderungen festlegen. Wird ein Produkt für den anonymen Markt entwickelt, dann werden die Marketingabteilung und der Vertrieb die wichtigen Anforderungen bestimmen. Je nach Situation ist also »man« festzulegen.

H. Balzert, *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*,
DOI 10.1007/978-3-8274-2247-7_16, © Spektrum Akademischer Verlag Heidelberg 2009

Was sind »Eigenschaften«?

Frage Welche Eigenschaften müssen Ihrer Meinung nach für ein neues Softwaresystem festgelegt werden? Versuchen Sie eine Klassifikation.

Antwort Visionen und Ziele, die mit dem Produkt erreicht werden sollen, stehen oft am Anfang einer Produktspezifikation:

■ »Visionen und Ziele«, S. 456

Rahmenbedingungen legen Restriktionen für das zu entwickelnde System oder den Entwicklungsprozess fest:

■ »Rahmenbedingungen«, S. 459

Die Umgebung, in die das zu entwickelnde System eingebettet wird, muss definiert werden:

■ »Kontext und Überblick«, S. 461

Die eigentlichen Eigenschaften eines Softwaresystems werden in der Regel in **funktionale** und **nichtfunktionale** Anforderungen unterteilt.

Eine funktionale Anforderung lässt sich wie folgt definieren:

Definition

> Eine **funktionale Anforderung** legt eine vom Softwaresystem oder einer seiner Komponenten bereitzustellende Funktion oder bereitzustellenden Service fest.

Funktionale Anforderungen lassen sich gliedern in (siehe »Basiskonzepte«, S. 99):

- ☐ Anforderungen, die die **Statik** des Systems beschreiben,
- ☐ Anforderungen, die die **Dynamik** des Systems beschreiben und
- ☐ Anforderungen, die die **Logik** des Systems beschreiben.

Unter einer nichtfunktionalen Anforderung versteht man oft alles, was *keine* funktionale Anforderung ist, was aber nicht ganz stimmt:

■ »Nichtfunktionale Anforderungen«, S. 463

Nichtfunktionale Anforderungen lassen sich unter dem Begriff Qualitätsanforderungen subsumieren:

■ »Box: Qualitätsmerkmale nach ISO/IEC 9126–1«, S. 468

Bereits bei der Festlegung von Anforderungen muss überlegt werden, wie die Anforderungen auf Einhaltung überprüft werden sollen:

■ »Abnahmekriterien«, S. 471

TBD Anforderungen, zu denen noch keine Festlegungen getroffen werden können, werden oft mit TBD gekennzeichnet. TBD steht für *to be defined* (noch nicht definiert) oder *to be determined* (noch nicht festgelegt).

16.1 Visionen und Ziele

Bevor funktionale und nichtfunktionale Anforderungen sowie Rahmenbedingungen festgelegt werden, sollten die Visionen und Ziele des Systems aufgestellt werden.

Warum sollten Visionen und Ziele vorher definiert werden? Frage

Sind Visionen und Ziele beschrieben, dann können funktionale und nichtfunktionale Anforderungen sowie die Rahmenbedingungen immer gegen die Visionen und Ziele abgeglichen werden. Es kann dann immer gefragt werden: »Ist diese Anforderung zielführend, d. h., trägt sie dazu bei, das Ziel zu erreichen?« Antwort

Vision

Eine **Vision** ist eine realitätsnahe Vorstellung der gewünschten Zukunft. Sie beschreibt, was erreicht werden soll, sagt aber nicht wie.

Beispiele 1

- Die Brüder Albrecht hatten folgende Vision: »Wir verkaufen qualitativ hochwertige Produkte möglichst preiswert.« Die Umsetzung dieser Vision führte zu den ALDI-Läden (*Al*brecht *Di*scount).
- Artur Fischer, der Erfinder des Dübel, hatte die Vision: »Ein Fotograf kann auch bei Nacht fotografieren.« Diese Vision brachte ihn dazu, das Blitzlicht zu erfinden.
- Die Paketverteilfirma Federal Express formulierte in ihrer Anfangszeit folgende Vision: »Wir liefern das Paket am nächsten Morgen bis 10.30 Uhr aus«.

Bezogen auf die Softwaretechnik sollte eine Vision als Leitgedanke für alle *Stakeholder* dienen.

Beispiel: SemOrg

/V10/ Die Firma Teachware soll durch das System in die Lage versetzt werden, die von ihr veranstalteten Seminare sowie Kunden und Dozenten effizient rechnerunterstützt zu verwalten.
/V20/ Die Kunden der Firma Teachware sollen über das Web möglichst viele Vorgänge selbst durchführen können.

Beispiel: Fensterheber

/V20/ Die Fensterheber-Komponente soll das komfortable Heben und Senken der Seitenfenster des Fahrzeugs ermöglichen.

Ziele

Ausgehend von einer Vision dienen **Ziele** dazu, die Vision zu verfeinern und zu operationalisieren.

Beispiel: SemOrg

/Z10/ Ein Interessent oder ein Kunde kann mindestens 20 Stunden jeden Tag Seminare und Veranstaltungen über das Web selektieren und eine Veranstaltung online buchen, damit die Mitarbeiter der Fa. Teachware von solchen Tätigkeiten entlastet werden.

In [Pohl07, S. 100 ff.] werden sieben Regeln zur Formulierung von Zielen aufgeführt:

- **Regel 1: Ziele kurz und prägnant formulieren**
  Füllwörter und Allgemeinplätze vermeiden.

Beispiel: SemOrg
Falsch: Es wäre gut, wenn zu Seminarveranstaltungen vielfältige Auswertungen möglich wären.
Besser: Das System soll dem Benutzer die Möglichkeit bieten, Seminarveranstaltungen nach den vorhandenen Attributen zu selektieren und zu sortieren.

- **Regel 2: Aktivformulierungen verwenden**
  Den Akteur klar benennen.

Beispiel
»Federal Express liefert das Paket aus« (siehe Beispiele 1).

- **Regel 3: Überprüfbare Ziele formulieren**
  Das Zielerfüllung muss im späteren System überprüfbar sein.

Beispiel
»Federal Express liefert bis 10.30 Uhr aus« (siehe Beispiele 1).

- **Regel 4: Ziele, die nicht überprüft werden können, nicht verfeinern**
  Ziel in überprüfbare Teilziele aufgliedern.

Beispiel
»Wir können jederzeit feststellen, wo sich ein Paket zurzeit befindet.«
Besser: »Wir können zwischen 7.00 Uhr und 20.00 Uhr innerhalb von maximal 5 Minuten feststellen, wo sich ein Paket zurzeit befindet.«
»Wir können zwischen 20.00 Uhr und 7.00 Uhr innerhalb von maximal 10 Minuten feststellen, wo sich ein Paket zurzeit befindet.«

- **Regel 5: Den Mehrwert eines Ziels hervorheben**
  Möglichst genau beschreiben, welchen Mehrwert das Ziel bringt.

Beispiel
Die Erstellungszeit von Rechnungen soll von 5 auf 2 Arbeitstage verkürzt werden.

- **Regel 6: Das Ziel sollte begründet werden**
  Eine Zielbegründung führt zur Identifikation weiterer Ziele.

Beispiel
Als Webanwendung kann das System auch durch externe Benutzer benutzt werden.

- **Regel 7: Keine Lösungsansätze angeben**
  Lösungsansätze schränken den Lösungsraum zu früh ein, daher den maximalen Lösungsraum offen lassen.

Beispiel: SemOrg
Falsch: Die Software »Seminarorganisation« muss mit der Software »Buchhaltung« permanent über eine Netzverbindung verbunden sein.
Richtig: Die Software »Seminarorganisation« muss mit der Software »Buchhaltung« in der Regel innerhalb von 24 Stunden einmal Daten austauschen können, mindestens aber einmal innerhalb von 5 Arbeitstagen. Begründung: Aus fachlicher Sicht reicht diese Festlegung, da nach einer Seminaranmeldung nicht sofort eine Mitteilung

an die Buchhaltung erfolgen muss. Außerdem ist eine Festlegung auf eine Netzverbindung nicht nötig. Es könnte ja auch sein, dass beide Programme auf einem Computersystem laufen.

In [Rupp07, S. 100 f.] werden zwei weitere Regeln aufgeführt:

- **Regel 8: Einschränkende Rahmenbedingungen aufführen**
  Rahmenbedingungen, die den Lösungsraum einschränken, müssen aufgeführt werden, damit die spätere Lösung nicht außerhalb des Lösungsraums liegt.

Wenn der Kollisionssensor anschlägt, muss das System in weniger als 20 Millisekunden den Airbag auslösen. Beispiel

- **Regel 9: Realistische Ziele formulieren**
  Unrealistische, nicht erreichbare Ziele diskreditieren die gesamten Anforderungen und werden nicht ernst genommen.

Die Spracheingabe muss alle deutschen Dialekte ohne Trainingsphase 100 %ig erkennen. Beispiel

In [NuEa07] werden die Probleme beschrieben, die durch den inkonsistenten Gebrauch der Terminologie auftreten, wenn *Stakeholder* zielorientierte Anforderungen formulieren. Es wird zwischen harten Zielen und weichen Zielen unterschieden. Es wird eine Methode vorgestellt, um diese Probleme zu lösen. Probleme

Weiterführende Konzepte zu Visionen und Zielen finden Sie in [Pohl07, S. 89 ff.] Literatur

16.2 Rahmenbedingungen

Eine **Rahmenbedingung** *(constraint)* – auch Restriktion genannt – legt organisatorische und/oder technische Restriktionen für das Softwaresystem und/oder den Entwicklungsprozess fest. Definition

Zu den **organisatorischen Rahmenbedingungen** gehören:

- **Anwendungsbereiche**: Legt fest, für welche Anwendungsbereiche das System vorgesehen ist, z. B. Textverarbeitung im Büro.
- **Zielgruppen**: Legt fest, für welche Zielgruppen das System vorgesehen ist, z. B. Sekretärinnen, Schreibkräfte. Diese Rahmenbedingung ist wichtig für die Gestaltung der Benutzungsoberfläche. Unter Umständen sollte auch festgelegt werden, von welchen Voraussetzungen, z. B. bezüglich des Qualifikationsniveaus des Benutzers, ausgegangen wird. Ebenfalls kann es sinnvoll sein, explizit anzugeben, für welche Anwendungsbereiche und Zielgruppen das Produkt *nicht* vorgesehen ist, z. B. für den IT-unkundigen Benutzer.

- **Betriebsbedingungen**: Folgende Punkte werden beschrieben:
  - Physikalische Umgebung des Systems, z. B. Büroumgebung, Produktionsanlage oder mobiler Einsatz.
  - Tägliche Betriebszeit, z. B. Dauerbetrieb bei Telekommunikationsanlagen.
  - Ständige Beobachtung des Systems durch Bediener oder unbeaufsichtigter Betrieb.

Deckt das System verschiedene Anwendungsbereiche und Zielgruppen ab, dann ist eine Aufführung der unterschiedlichen Bedürfnisse und Anforderungen nötig.

Beispiel: SemOrg

- **Anwendungsbereich**: Kaufmännisch/administrativer Anwendungsbereich
- **Zielgruppen**:
  Mitarbeiter der Firma Teachware lassen sich gliedern in:
  Kundensachbearbeiter, Seminarsachbearbeiter, Veranstaltungsbetreuer.
  Kunden der Firma Teachware:
  Kunden und Firmen können sich über das Internet über Seminare und Veranstaltungen informieren und selbst Buchungen durchführen.
- **Betriebsbedingungen**: Büroumgebung

Zu den **technischen Rahmenbedingungen** gehören:

- Technische Produktumgebung
- Anforderungen an die Entwicklungsumgebung

Bei der **technischen Produktumgebung** sind folgende Festlegungen zu treffen:

- **Software**: Welche Softwaresysteme (Betriebssystem, Laufzeitsystem, Datenbank, Fenstersystem usw.) sollen auf der Zielmaschine (Maschine, auf der das fertiggestellte System eingesetzt werden soll) zur Verfügung stehen, z. B. Web-Browser auf dem Client.
- **Hardware**: Welche Hardware-Komponenten (CPU, Peripherie, z. B. Grafikbildschirm, Drucker) sind in minimaler und maximaler Konfiguration für den Systemeinsatz vorgesehen.
- **Orgware**: Unter welchen organisatorischen Randbedingungen bzw. Voraussetzungen soll das Produkt eingesetzt werden (z. B. »Aktuelle Wetterdaten sind nur dann einbindbar, wenn das System über einen permanenten Internet-Zugriff verfügt«).

Bei Client/Server- oder Web-Anwendungen ist die Umgebung jeweils für Clients und Server getrennt anzugeben.

Bei den **Anforderungen an die Entwicklungsumgebung** sind folgende Festlegungen zu treffen:

- Software
- Hardware
- Orgware

- Entwicklungsschnittstellen

Es wird die Entwicklungsumgebung des Systems beschrieben. Es wird festgelegt, welche Konfiguration bzgl. Software, Hardware und Orgware für die Entwicklung des Systems benötigt wird. Diese Festlegungen sind insbesondere dann notwendig, wenn Entwicklungs- und Zielmaschine unterschiedlich sind. Bei Entwicklungsschnittstellen ist unter Umständen aufzuführen, über welche einzuhaltenden Hardware- und Softwareschnittstellen Entwicklungs- und Zielrechner gekoppelt sind. Unter Software ist insbesondere aufzuführen, welche Software-Werkzeuge, z. B. integrierte Entwicklungsumgebungen, Programmierumgebungen, Compiler usw., benötigt werden.

- **Technische Produktumgebung**: Das System ist eine Web-Anwendung.
  - **Software**: Server-Betriebssystem: Windows. Client: Web-Browser (Es werden die drei marktführenden Browser unterstützt).
  - Hardware: Server: PC, Client: Browserfähiges Gerät mit Grafikbildschirm
  - Orgware: Zugriff des Servers auf die Buchhaltungssoftware.
- **Anforderungen an die Entwicklungsumgebung**: Keine Abweichungen von der Einsatzumgebung.

Beispiel: SemOrg

Rahmenbedingungen können

- keine der Anforderungen einschränken,
- mögliche Realisierungen von Anforderungen einschränken,
- zur Änderung von Anforderungen führen,
- zu neuen Anforderungen führen oder
- zu nicht realisierbaren Anforderungen führen.

16.3 Kontext und Überblick

Jedes Softwaresystem ist in eine materielle und immaterielle Umgebung eingebettet – mehr oder weniger stark in Abhängigkeit von der Art der Software.

Materielle Umgebung: Sensoren, Gebäude, Personen, andere technische Systeme, physikalische Kanäle und Übertragungsmedien.
Immaterielle Umgebung: Schnittstellen zu anderen Softwaresystemen, Internet.

Beispiele

Die Systemumgebung hat einen maßgeblichen Einfluss auf die Anforderungen. Daher ist es wichtig, die Systemumgebung festzulegen. Die Abb. 16.3-1 zeigt, wie ein System und seine Umgebung abgegrenzt werden können (in Anlehnung an [Pohl07, S. 56 ff.]).

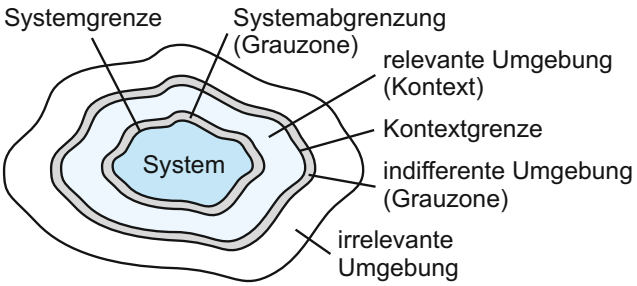


Abb. 16.3-1: Das System und seine Umgebung.

Systemgrenze

Das zu entwickelnde System besitzt eine **Systemgrenze**, die es von den Teilen der Umgebung abgrenzt, die durch die Entwicklung *nicht* verändert werden können. Liegt bereits fest, was *nicht* zum System gehört, dann sollte dies explizit aufgeführt werden.

Quellen und Senken interagieren mit dem System. Quellen liefern Eingaben, Senken erhalten Ausgaben. Beispiele für Quellen und Senken sind Personen, andere Systeme, Sensoren und Aktoren. Die Interaktion erfolgt i. Allg. über Benutzungsschnittstellen und Softwareschnittstellen.

Da am Anfang des *Requirements Engineering* die Systemgrenze noch nicht genau festliegt, wird sie von einer Grauzone umgeben.

Beispiel: SemOrg

Das System SemOrg soll mit der Buchhaltungssoftware Daten austauschen. Da zum Zeitpunkt der Anforderungserstellung das Buchhaltungssystem noch nicht feststeht, kann auch die Schnittstelle noch nicht fertig spezifiziert werden.

Kontext

Um das System herum gibt es eine relevante Umgebung – **Kontext** genannt –, die für die Systementwicklung zu beachten ist, und eine irrelevante Umgebung, die keinen Einfluss auf die Entwicklung hat. Zwischen beiden gibt es ebenfalls eine Grauzone.

Der Kontext beeinflusst die Interpretation einer Anforderung, daher ist die Festlegung des Kontextes so wichtig.

Beispiel

Die Reaktion des Systems auf Benutzereingaben muss in weniger als 2 Sekunden erfolgen. Legt der Kontext fest, dass die Anwendung auf einem Handy läuft und die Internet-Verbindung über das Handy erfolgt, dann hat die Anforderung andere Konsequenzen, als wenn die Anwendung auf einem Notebook oder PC läuft und über einen DSL-Anschluss verfügt.

Basieren Anforderungen auf Annahmen des Systemkontextes, dann sollten diese explizit gekennzeichnet werden.

Notation

Neben der natürlichsprachlichen Beschreibung des Kontextes eignen sich auch folgende Basiskonzepte zur Beschreibung:

- UML-Use-Case-Diagramm (siehe »*Use Case*-Diagramme und -Schablonen«, S. 255)
- UML-Klassendiagramm (siehe »Zusammenfassung von Funktionen«, S. 131)
- UML-Sequenzdiagramm (siehe »Sequenzdiagramm«, S. 333)

- UML-Verteilungsdiagramm (für physikalische Kanäle)

Den Kontext zur Seminarorganisation wird in der Abb. 16.3-2 durch ein *Use Case*-Diagramm dargestellt – beispielsweise im Lastenheft. Eine Verfeinerung – beispielsweise im Pflichtenheft – zeigt die Abb. 16.3-3.

Beispiel: SemOrg

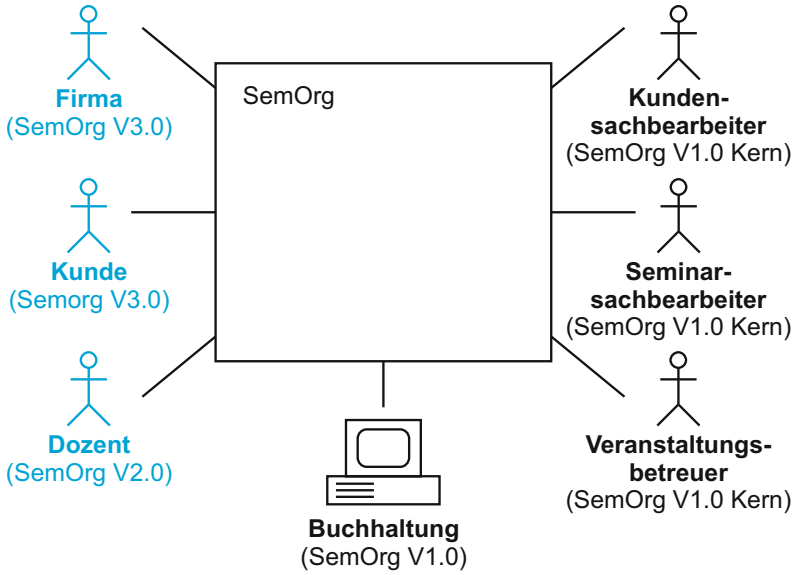


Abb. 16.3-2: Umwelt des Produkts SemOrg (Umweltdiagramm) als UML-Use-Case-Diagramm.

Weiterführend wird dieses Thema in [Pohl07, S. 55 ff.] behandelt. Literatur

16.4 Nichtfunktionale Anforderungen

Unter dem Begriff nichtfunktionale Anforderungen wird oft alles subsumiert, was nicht zu den funktionalen Anforderungen gehört.

Was verstehen Sie unter nichtfunktionalen Anforderungen? Frage

Die Auffassungen, was nichtfunktionale Anforderungen sind, differieren. Antwort

Nichtfunktionale Anforderungen (*nonfunctional requirements*, kurz NFRs), auch Technische Anforderungen oder *Quality of Service* (QoS) genannt, beschreiben Aspekte, die typischerweise mehrere oder alle funktionale Anforderungen betreffen bzw. überschneiden *(cross-cut)*. Zu den NFRs zählen u. a. Genauigkeit, Verfügbarkeit, Nebenläufigkeit, Konsumierbarkeit (eine Obermenge der Benutzbarkeit), Internationalisierung, Betriebseigenschaften, Zuverlässigkeit, Sicherheit, Service-Anforderungen, Support (siehe auch [Ambl08, S. 64]). Nichtfunktionale Anforderungen haben großen Einfluss auf die Softwarearchitektur.

Außerdem beeinflussen sich nichtfunktionale Anforderungen. Sicherheit *(security)* steht häufig im Konflikt mit Benutzbarkeit, Speichereffizienz ist meist gegenläufig zur Laufzeiteffizienz.

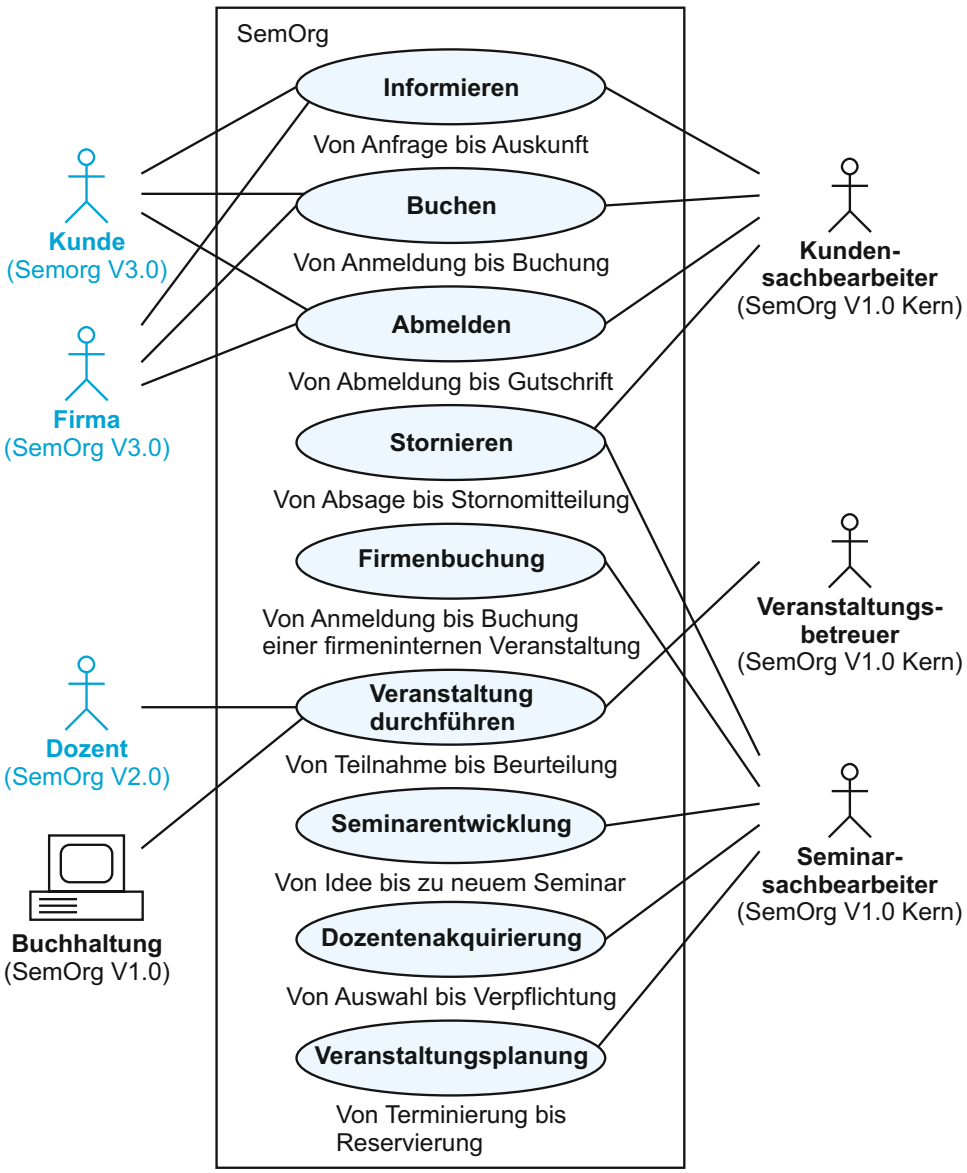


Abb. 16.3-3: UML-Use Cases des Produkts SemOrg (Übersichtsdiagramm).

In [Pohl07, S. 16 ff.] wird darauf hingewiesen, dass nichtfunktionale Anforderungen entweder Qualitätsanforderungen oder unterspezifizierte funktionale Anforderungen sind.

Definition

Eine **Qualitätsanforderung** legt eine qualitative und/oder quantitative Eigenschaft des Softwaresystems oder einer seiner Komponenten fest.

Qualitätsanforderungen legen fest, welche Qualitätsmerkmale für das Softwaresystem als relevant erachtet werden und in welcher Qualitätsstufe sie erreicht werden sollen. Qualitätsanforderungen haben in der Regel Auswirkungen auf den Softwareentwurf, da sie aufgrund ihrer systemübergreifenden Eigenschaft die Struktur einer Architektur wesentlich beeinflussen.

Unterspezifizierte Anforderungen lassen sich bei entsprechender Detaillierung in funktionale Anforderungen und ggf. Qualitätsanforderungen überführen.

Allgemein legen funktionale Anforderungen fest, »was« ein System tut bzw. tun soll, während die Qualitätsanforderungen beschreiben, »wie gut« die Funktionen erledigt werden (sollen). In [Glin08] wird folgende Klassifikation der Anforderungen vorgeschlagen:

- **Funktionale Anforderungen**: Funktionalität und Verhalten
  - Funktionen
  - Daten
  - Stimuli
  - Reaktionen
  - Verhalten
- **Leistungsanforderungen**: Zeit- und Speichergrenzen
  - Zeit
  - Geschwindigkeit
  - Umfang
  - Durchsatz
- **Spezifische Qualitätsanforderungen**: »-keiten«
  - Zuverlässigkeit
  - Benutzbarkeit
  - Sicherheit
  - Verfügbarkeit
  - Portabilität
  - Wartbarkeit usw.
- **Randbedingungen/Einschränkungen**
  - Physikalisch
  - Rechtlich
  - Kulturell
  - Umgebung
  - Entwurf und Implementierung
  - Schnittstellen usw.

ISO/IEC 9126–1

In der internationalen Norm ISO/IEC 9126-1:2001 »*Software engineering – Product quality – Part 1: Quality model*« wird ein Qualitäts-Modell beschrieben, das aus sechs Qualitätsmerkmalen *(characteristics)* besteht, die wiederum in insgesamt 26 Teilmerkmale *(subcharacteristics)* untergliedert sind. Den Merkmalen bzw. Teilmerkmalen sind Qualitätsattribute *(quality attributes)* zugeordnet. Alle Qualitätsmerkmale sind auf jede Art von Software anwendbar.

6 Q-Merkmale

Die sechs Qualitätsmerkmale sind:

- **Funktionalität** *(functionality)*
- **Zuverlässigkeit** *(reliability)*
- **Benutzbarkeit** *(usability)*
- **Effizienz** *(efficiency)*
- **Wartbarkeit** *(maintainability)*

- **Portabilität** *(portability)*

Die Definition dieser Qualitätsmerkmale sowie aller Teilmerkmale sind im Kapitel »Box: Qualitätsmerkmale nach ISO/IEC 9126–1«, S. 468, aufgeführt. Anhand dieser Qualitätsmerkmale können die nichtfunktionalen Anforderungen gegliedert aufgeführt werden. Den Qualitätsmerkmalen kann man Qualitätsstufen zuordnen, z. B.:

- Sehr gute Systemqualität
- Gute Systemqualität
- Normale Systemqualität

Beispiel 1a: SemOrg

Im Lastenheft wird die in der Tab. 16.4-1 angegebene Qualitätszielbestimmung vorgenommen.

Tab. 16.4-1: Qualitätszielbestimmung für SemOrg im Lastenheft.

Systemqualität	sehr gut	gut	normal	nicht relevant
Funktionalität		X		
Zuverlässigkeit			X	
Benutzbarkeit		X		
Effizienz			X	
Wartbarkeit			X	
Portabilität			X	

In einem Pflichtenheft kann eine Qualitätszielbestimmung für alle 26 Teilmerkmale vorgenommen werden.

Beispiel 1b: SemOrg

Im Pflichtenheft wird eine detaillierte Qualitätszielbestimmung vorgenommen, siehe »Fallstudie: SemOrg – Die Spezifikation«, S. 107.

Spezifikation von nichtfunktionalen Anforderungen

Nichtfunktionale Anforderungen werden in der Regel in natürlicher Sprache formuliert (siehe auch »Natürlichsprachliche Anforderungen«, S. 481).

Beispiel 1c: SemOrg

/QEZ10/ Alle Reaktionszeiten auf Benutzeraktionen müssen unter 5 Sekunden liegen (EZ steht für **E**ffizienz / **Z**eitverhalten).

In der UML können nichtfunktionale Anforderungen – über das Modell verstreut – in Form von Zusicherungen an Modellelemente spezifiziert werden (siehe auch »Constraints und die OCL in der UML«, S. 377). Diese Art und Weise, nichtfunktionale Anforderungen zu spezifizieren, ist aber unübersichtlich. Eine Alternative dazu bietet die Sprache **SysML** (siehe »Basiskonzepte«, S. 99). In ihr gibt es das Modellelement *Requirement* und die neue Diagrammart *Requirement Diagram*. Dieses Modellelement wird als stereotypisierte UML-Klasse mit dem Stereotyp `«requirement»` und den Stereotypattributen `id` und `text` dargestellt. `id` legt eine eindeutige Kennung fest,

`text` einen beschreibenden Text, der auch Verweise auf Quellen außerhalb des UML-Modells enthalten kann. Damit können funktionale und nichtfunktionale Anforderungen spezifiziert werden.

Da es sich bei dem Modellelement *Requirement* um eine Klasse handelt, sind auch alle Klassenbeziehungen erlaubt. Folgende Abhängigkeitsbeziehungen (gestrichelter Pfeil in der UML und in SysML) können modelliert werden:

- «`trace`»: Die Anforderung A bezieht sich auf die Anforderung B. Sie kann benutzt werden, um den Zusammenhang zwischen funktionalen und nichtfunktionalen Anforderungen zu modellieren.
- «`deriveReqt`»: Die Anforderung A ist von der Anforderung B abgeleitet. «`deriveReqt`» ist ein Stereotyp der UML-Abstraktionsbeziehung.
- «`verify`»: Das Testelement A überprüft die Anforderung B. «`verify`» ist eine Spezialisierung des UML-Stereotyps «`trace`».

Die Abb. 16.4-1 zeigt die grafische Spezifikation von nichtfunktionalen Anforderungen.

Beispiel 1d: SemOrg

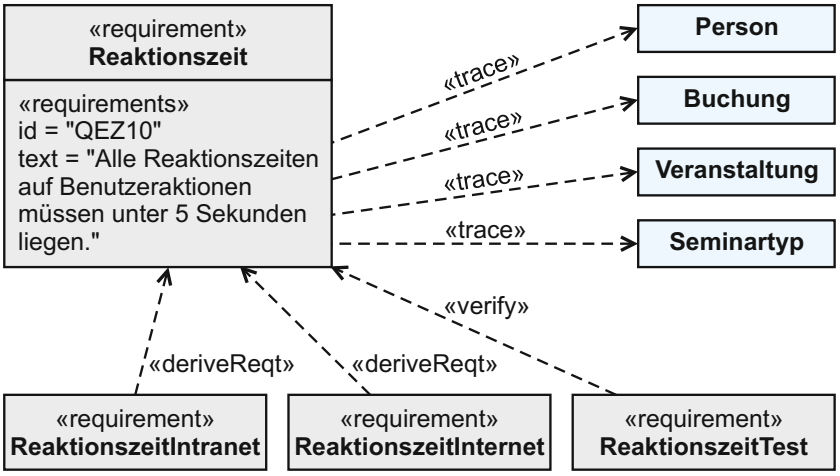


Abb. 16.4-1: Beispiel für die Spezifizierung nichtfunktionaler Anforderungen in der UML 2 und der SysML.

Neben speziellen SysML-Werkzeugen können diese Modellelemente mit jedem UML-Werkzeug, das Stereotypen mit Eigenschaften (Stereotypattribute) unterstützt, umgesetzt werden.

Werkzeuge

[Weil06], [FMS08], [JaMa02], [OBN+08].

Weiterführende Literatur

16.5 Box: Qualitätsmerkmale nach ISO/IEC 9126–1

Funktionalität *(functionality)*

Fähigkeit des Softwareprodukts, Funktionen bereitzustellen, damit die Software unter den spezifizierten Bedingungen die festgelegten Bedürfnisse erfüllt.

- **Angemessenheit** *(suitability)*
  Fähigkeit des Softwareprodukts, für spezifizierte Aufgaben und Benutzerziele geeignete Funktionen bereitzustellen.
- **Genauigkeit** *(accuracy)*
  Fähigkeit des Softwareprodukts, die richtigen oder vereinbarten Ergebnisse oder Wirkungen mit der benötigten Genauigkeit bereitzustellen.
- **Interoperabilität** *(interoperability)*
  Fähigkeit des Softwareprodukts, mit einem oder mehreren vorgegebenen Systemen zusammenzuwirken.
- **Sicherheit** *(security)*
  Fähigkeit des Softwareprodukts, Informationen und Daten so zu schützen, dass nicht autorisierte Personen oder Systeme sie nicht lesen oder verändern können, und autorisierten Personen oder Systemen der Zugriff nicht verweigert wird.
- **Konformität der Funktionalität** *(functionality compliance)*
  Fähigkeit des Softwareprodukts, Standards, Konventionen oder gesetzliche Bestimmungen und ähnliche Vorschriften bezogen auf die Funktionalität einzuhalten.

Zuverlässigkeit *(reliability)*

Fähigkeit des Softwareprodukts, ein spezifiziertes Leistungsniveau zu bewahren, wenn es unter festgelegten Bedingungen benutzt wird.

- **Reife** *(maturity)*
  Fähigkeit des Softwareprodukts, trotz Fehlzuständen in der Software nicht zu versagen.
- **Fehlertoleranz** *(fault tolerance)*
  Fähigkeit des Softwareprodukts, ein spezifiziertes Leistungsniveau bei Software-Fehlern oder Nicht-Einhaltung ihrer spezifizierten Schnittstelle zu bewahren.
- **Wiederherstellbarkeit** *(recoverability)*
  Fähigkeit des Softwareprodukts, bei einem Versagen ein spezifiziertes Leistungsniveau wiederherzustellen und die direkt betroffenen Daten wiederzugewinnen.
- **Konformität der Zuverlässigkeit** *(reliability compliance)*
  Fähigkeit des Softwareprodukts, Standards, Konventionen oder Vorschriften bezogen auf die Zuverlässigkeit einzuhalten.

Benutzbarkeit *(usability)*

Fähigkeit des Softwareprodukts, vom Benutzer verstanden und benutzt zu werden sowie für den Benutzer erlernbar und »attraktiv« zu sein, wenn es unter den festgelegten Bedingungen benutzt wird.

- **Verständlichkeit** *(understandability)*
  Fähigkeit des Softwareprodukts, den Benutzer zu befähigen, zu prüfen, ob die Software angemessen ist und wie sie für bestimmte Aufgaben und Nutzungsbedingungen verwendet werden kann.
- **Erlernbarkeit** *(learnability)*
  Fähigkeit des Softwareprodukts, den Benutzer zu befähigen, die Anwendung zu erlernen.
- **Bedienbarkeit** *(operability)*
  Fähigkeit des Softwareprodukts, den Benutzer zu befähigen, die Anwendung zu bedienen und zu steuern.
- **Attraktivität** *(attractiveness)*
  Fähigkeit des Softwareprodukts für den Benutzer attraktiv zu sein, z. B. durch die Verwendung von Farbe oder die Art des grafischen Designs.
- **Konformität der Benutzbarkeit** *(usability compliance)*
  Fähigkeit des Softwareprodukts, Standards, Konventionen, Stilvorgaben *(style guides)* oder Vorschriften bezogen auf die Benutzbarkeit einzuhalten.

Effizienz *(efficiency)*

Fähigkeit des Softwareprodukts, ein angemessenes Leistungsniveau bezogen auf die eingesetzten Ressourcen unter festgelegten Bedingungen bereitzustellen.

- **Zeitverhalten** *(time behaviour)*
  Fähigkeit des Softwareprodukts, angemessene Antwort- und Verarbeitungszeiten sowie Durchsatz bei der Funktionsausführung unter festgelegten Bedingungen sicherzustellen.
- **Verbrauchsverhalten** *(resource utilisation)*
  Fähigkeit des Softwareprodukts, eine angemessene Anzahl und angemessene Typen von Ressourcen zu verwenden, wenn die Software ihre Funktionen unter festgelegten Bedingungen ausführt.
- **Konformität der Effizienz** *(efficiency compliance)*
  Fähigkeit des Softwareprodukts, Standards oder Konventionen bezogen auf die Effizienz einzuhalten.

Wartbarkeit *(maintainability)*

Fähigkeit des Softwareprodukts änderungsfähig zu sein. Änderungen können Korrekturen, Verbesserungen oder Anpassungen der Software an Änderungen der Umgebung, der Anforderungen und der funktionalen Spezifikationen einschließen.

- **Analysierbarkeit** *(analyzability)*
  Fähigkeit des Softwareprodukts, Mängel oder Ursachen von Versagen zu diagnostizieren oder änderungsbedürftige Teile zu identifizieren.
- **Änderbarkeit** *(changeability)*
  Fähigkeit des Softwareprodukts, die Implementierung einer spezifizierten Änderung zu ermöglichen.
- **Stabilität** *(stability)*
  Fähigkeit des Softwareprodukts, unerwartete Wirkungen von Änderungen der Software zu vermeiden.
- **Testbarkeit** *(testability)*
  Fähigkeit des Softwareprodukts, die modifizierte Software zu validieren.
- **Konformität der Wartbarkeit** *(maintainability compliance)*
  Fähigkeit des Softwareprodukts, Standards oder Konventionen bezogen auf die Wartbarkeit einzuhalten.

Portabilität *(portability)*

Fähigkeit des Softwareprodukts, von einer Umgebung in eine andere übertragen zu werden. Umgebung kann organisatorische Umgebung, Hardware- oder Software-Umgebung einschließen.

- **Anpassbarkeit** *(adaptability)*
  Fähigkeit des Softwareprodukts, die Software an verschiedene, festgelegte Umgebungen anzupassen, wobei nur Aktionen oder Mittel eingesetzt werden, die für diesen Zweck für die betrachtete Software vorgesehen sind.
- **Installierbarkeit** *(installability)*
  Fähigkeit des Softwareprodukts, in einer festgelegten Umgebung installiert zu werden.
- **Koexistenz** *(co-existence)*
  Fähigkeit des Softwareprodukts, mit anderen unabhängigen Softwareprodukten in einer gemeinsamen Umgebung gemeinsame Ressourcen zu teilen.
- **Austauschbarkeit** *(replaceability)*
  Fähigkeit des Softwareprodukts, diese Software anstelle einer spezifizierten anderen in der Umgebung jener Software für denselben Zweck zu verwenden.

- **Konformität der Portabilität** *(portability compliance)*
  Fähigkeit des Softwareprodukts, Standards oder Konventionen bezogen auf die Portabilität einzuhalten.

16.6 Abnahmekriterien

Welche Vorteile bringt es mit sich, wenn mit der Festlegung der Anforderungen bereits die Festlegung von Abnahmekriterien für diese Anforderungen verbunden wird? Frage

Das bringt folgende Vorteile mit sich: Antwort

+ Es ist von vornherein klar, wie überprüft werden kann, ob die Anforderungen korrekt realisiert wurden.
+ Es wird sich bei der Abnahme nicht nur auf das realisierte System gestützt.
+ Bereits bei der Formulierung der Anforderungen wird darauf geachtet, dass sie auch überprüft werden können, was nach aller Erfahrung zu einer operationalisierten Festlegung und qualitativen Verbesserung der Anforderungen führt.
+ Die Formulierung von Abnahmekriterien führt zu besserer Veranschaulichung und Verständlichkeit der meist abstrakt formulierten Anforderungen.

Abnahmekriterien lassen sich nach ihrem **Abstraktionsgrad** klassifizieren: Klassifikation

- Abstrakte Abnahmekriterien enthalten keine konkreten Werte.
- Konkrete Abnahmekriterien enthalten konkrete Werte, die z. B. für einen Abnahmetest verwendet werden können.

Weiterhin lassen sich die Abnahmekriterien nach ihrer **Reichweite** klassifizieren:

- Abnahmekriterien gelten für genau eine Anforderung.
- Abnahmekriterien gelten für mehrere Anforderungen.
- Abnahmekriterien gelten nur für Teile einer Anforderung (Regelfall).

Analog, wie es Anforderungen an Anforderungen gibt, so gibt es auch (Qualitäts-)Anforderungen an Abnahmekriterien (vgl. [Rupp07, S. 326 f.]): Anforderungen

- Abnahmekriterien müssen *wirtschaftlich überprüfbar* sein, d. h. eine Prüfung, ob die Anforderung realisiert ist, muss mit vertretbarem Aufwand möglich sein.
- Abnahmekriterien müssen auf Korrektheit überprüfbar sein, d. h. eine Prüfung, ob die Anforderung *korrekt realisiert* ist, muss möglich sein.
- Abnahmekriterien müssen so formuliert sein, dass sie für **Regressionstests** verwendbar sind.

- Abnahmekriterien dürfen nicht mehr aber auch nicht weniger festlegen, als in der jeweiligen Anforderung gefordert wird. Es dürfen in einem Abnahmekriterium *keine* zusätzlichen Eigenschaften oder Leistungen versteckt sein, die in der Anforderung selbst nicht aufgeführt sind. Umgekehrt dürfen Details, die in Anforderungen stehen, bei Abnahmekriterien nicht weggelassen werden.
- Abnahmekriterien sollen *minimal*, aber *vollständig* sein, d. h. sie sollen alle Anforderungen überdecken, aber aus Wirtschaftlichkeitsgründen nicht mehrfach.

Notation

Abnahmekriterien lassen sich natürlichsprachlich, semiformal und formal beschreiben.

Natürlichsprachliche Abnahmekriterien können in drei Teile gegliedert werden:

- Ausgangssituation: Zustand des Prüflings vor der Abnahmeprüfung
- Ereignis: Welches Ereignis löst die Prüfung aus?
- Erwartetes Ergebnis: Soll-Zustand bei korrektem Verhalten.

Beispiel: SemOrg

Anforderung:
/F12/ Wenn ein Kunde oder eine Firma sich von einer bereits gebuchten Veranstaltung später als X Wochen vor der Veranstaltung abmeldet, dann muss das System Stornogebühren in Höhe der Veranstaltungsgebühr berechnen oder nach einem Ersatzteilnehmer fragen.
Abnahmekriterium (abstrakt):
Ausgangssituation: Ein Kunde, eine Firma, ein Seminar und eine Veranstaltung sind angelegt. Der Kunde und die Firma haben die Veranstaltung gebucht.
Ereignis: Der Kunde und die Firma stornieren die Veranstaltung X Wochen vor Veranstaltungsbeginn (wobei festgelegt sein muss, ab wann Stornogebühren in Höhe der Veranstaltungsgebühr anfallen, X muss in diesen Zeitraum fallen).
Erwartetes Ergebnis: Das System fragt, ob der Kunde und die Firma einen Ersatzteilnehmer stellen (Name usw. wird angefordert). Wenn nein, dann wird mitgeteilt, dass der volle Veranstaltungsbetrag fällig wird. Die Stornierung wird bestätigt.

Formulieren Sie das Abnahmekriterium des Beispiels in konkreter Form.

Sind Abnahmekriterien komplex, dann ergeben sich daraus oft viele natürlichsprachliche Abnahmekriterien.

Problem

Bei der Festlegung von Anforderungen wird oft nur der Standardfall beschrieben. Sonderfälle werden nicht spezifiziert und Fehlerfälle oft nicht betrachtet. Das hat zur Folge, dass bei der Realisierung des Systems die Entwickler oft nicht wissen, wie sie diese Fälle realisieren sollen.

Formalisierte Beschreibung

Als Alternativen für die Beschreibung von Abnahmekriterien bieten sich folgende Basiskonzepte an:

- »Entscheidungstabellen und Entscheidungsbäume«, S. 386
- »Geschäftsprozesse und *Use Cases*«, S. 250

Testverfahren

Für die möglichst umfassende, aber redundanzarme Prüfung der spezifizierten Funktionalität bieten sich **funktionale Testverfahren** (Black-Box-Tests) an, bei denen das zu testende System ein »schwarzer Kasten« ist, d. h. die interne Programmstruktur ist dem Tester nicht bekannt. Ein vollständiger Funktionstest ist i. Allg. nicht durchführbar. Ziel bei der Ermittlung der Abnahmekriterien muss es daher sein, Testfälle so auszuwählen, dass die Wahrscheinlichkeit groß ist, Fehler zu finden. Für die Testfallbestimmung gibt es folgende wichtige Verfahren:

- **Funktionale Äquivalenzklassenbildung**,
- **Grenzwertanalyse**,
- **Test spezieller Werte**,
- Zustandsbasierter Test,
- Ursache-Wirkungs-Analyse.

Nichtfunktionale Anforderungen

Auch – oder gerade – für nichtfunktionale Anforderungen sind Abnahmekriterien zu formulieren. Oft werden nichtfunktionale Anforderungen nur vage beschrieben. Die Formulierung von Abnahmekriterien hilft, die Anforderung zu operationalisieren.

Beispiel: SemOrg

Qualitätsanforderung:
/QEZ10/ Alle Reaktionszeiten auf Benutzeraktionen müssen unter 5 Sekunden liegen.
Abnahmekriterium:
Beim Aufruf der Funktionen /F10/, /F11/, /F12/, /F20/, /F30/, /F40/, /F50/, /F60/, /F61/ (siehe »Fallstudie: SemOrg – Die Spezifikation«, S. 107) muss die Ausgabe des Ergebnisses innerhalb von 5 Sekunden erfolgen.

Abnahmeszenarien

Die Prüfung von Geschäftsabläufen und *Use Cases* erfordern oft das Anwenden mehrerer Abnahmekriterien in einer bestimmten Reihenfolge: Ein **Abnahmeszenario** ist eine chronologisch geordnete Sammlung von Abnahmekriterien, das für eine Abnahme verwendet wird.

Beispiel: SemOrg

Ein Seminar neu erfassen, eine Veranstaltung neu erfassen, die Veranstaltung dem Seminar zuordnen, einen Dozenten erfassen und dem Seminar und der Veranstaltung zuordnen.

Es kann zwischen gültigen und ungültigen Abnahmeszenarien unterschieden werden. Ein gültiges Szenario führt zu einer korrekten Ausführung, ein ungültiges muss zu einem definierten Fehlerstatus führen.

Resümee

Die sorgfältige Formulierung von Abnahmekriterien ist aufwendig. Es werden dabei jedoch Lücken und Unvollständigkeiten aufgedeckt und Präzisierungen vorgenommen, die insgesamt zu einer wesentlichen Qualitätsverbesserung der Anforderungsspezifikation führen.

Zeitpunkt

Abnahmekriterien sollen zeitlich versetzt zur Festlegung der Anforderungen ermittelt werden, wenn eine gewisse Stabilisierung der Anforderungen eingetreten ist. Rückwirkungen, die sich aus den Abnahmekriterien auf die Anforderungen ergeben, können noch während der Analyse durch Änderung der betroffenen Anforderungen behoben werden.

Die Formulierung von Abnahmekriterien wird erleichtert, wenn jeweils fachlich zusammenhängende Anforderungspakete betrachtet werden.

Besitzen Anforderungen eine hohe Kritikalität (siehe »Anforderungsattribute«, S. 479), dann sind u. U. zusätzliche Abnahmekriterien zu formulieren, um sicher zu sein, dass die Anforderung exakt realisiert wird.

Die Abnahmekriterien sollten Bestandteil der Anforderungsspezifikation sein und damit auch Vertragsbestandteil zwischen Auftraggeber und Auftragnehmer.

Die Abnahmekriterien sollten von einer Person geschrieben werden, die Fachexperte in der betreffenden Domäne ist, aber die Anforderungen *nicht* erstellt hat. Die Überprüfung der Abnahmekriterien wiederum sollte durch diejenigen erfolgen, die die Anforderungen erstellt haben.

Überlegen Sie, warum dies sinnvoll ist.

17 Anforderungen an Anforderungen

Frage

Welche Anforderungen würden Sie an jede einzelne Anforderung stellen?

Antwort

Jede *einzelne* Anforderung soll eine Reihe von Kriterien erfüllen (in Anlehnung an [IEEE830, S. 4 ff.]). Sie soll sein:

- **Korrekt** *(correct)*: Die Anforderung ist korrekt, wenn das zu erstellende Softwaresystem sie erfüllen soll. Der Auftraggeber und/oder die Benutzer können sagen, ob die Anforderung die Bedürfnisse reflektiert.
- **Eindeutig** *(unambiguous)*: Die Anforderung wird von allen *Stakeholdern* gleich interpretiert. Als Minimum sollte die beschriebene Charakteristik durch einen einzigen, eindeutigen Begriff beschrieben werden. Wird ein Begriff in verschiedenen Kontexten unterschiedlich verwendet, dann sollte er in einem Glossar definiert werden. Alle Erfahrungen haben gezeigt, dass in natürlicher Sprache beschriebene Anforderungen meist nicht eindeutig interpretiert werden. Das hat zur Entwicklung formalisierter Beschreibungsverfahren geführt.
- **Vollständig** *(complete)*: Die Anforderung muss die geforderte Funktionalität vollständig beschreiben. Ist die Anforderung noch unvollständig, dann ist dies zu kennzeichnen, z. B. durch `TBD` *(to be determined)*.
- **Konsistent** *(consistent)*: Die Anforderung muss in sich widerspruchsfrei sein.
- **Klassifizierbar nach Wichtigkeit** *(ranked for importance)*: Der Anforderung kann eine Wichtigkeitsstufe zugeordnet werden. Einige Anforderungen können essenziell sein, z. B. bei lebenskritischen Anwendungen, während andere nur wünschenswert sind. Mögliche Klassen sind:
  - ☐ Essenziell *(essential)*: Das Softwaresystem kann ohne die Erfüllung der Anforderung nicht eingesetzt werden.
  - ☐ Bedingt *(conditional)*: Die Anforderung wertet das Softwaresystem auf, aber die Nichtrealisierung macht das System nicht unbrauchbar.
  - ☐ Optional *(optional)*: Anforderung, die das System u.U wertvoller macht. Gibt dem Auftragnehmer die Möglichkeit, etwas Zusätzliches dem Auftraggeber anzubieten.

H. Balzert, *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*,
DOI 10.1007/978-3-8274-2247-7_17, © Spektrum Akademischer Verlag Heidelberg 2009

- **Klassifizierbar nach Stabilität** *(ranked for stability)*: Der Anforderung kann eine Stabilitätsstufe zugeordnet werden. Unter Stabilität wird hierbei die Anzahl der erwarteten Änderungen bei dieser Anforderung verstanden. Dieser Wert kann ein Erfahrungswert sein oder das Wissen über vorauszusehende Veränderungen, die die Organisation, die Funktionen oder die Personen, die durch die Software unterstützt werden, betreffen.
- **Überprüfbar** *(verifiable)*: Die Anforderung muss so beschrieben sein, dass sie nach der Realisierung überprüfbar bzw. testbar ist. Es muss ein kosteneffektives Verfahren vorhanden sein, mit der eine Person oder eine Maschine überprüfen kann, dass das System die Anforderung erfüllt. Jede inkonsistente Anforderung ist nicht überprüfbar. Nicht überprüfbare Anforderungen enthalten Aussagen wie »Funktioniert gut«, »gute Benutzungsoberfläche«, »soll normalerweise eintreten«.
- **Verfolgbar** *(traceable)*: Die Anforderung muss sich eindeutig identifizieren lassen, z. B. durch eine eindeutige Anforderungsnummer, die unverändert bleibt.

Frage Welche Anforderungen soll eine Anforderungsspezifikation, die alle Anforderungen zu einem Softwaresystem enthält, erfüllen?

Antwort Alle Anforderungen zusammen sollen sein (teilweise nach [IEEE830, S. 4 ff.]):

- **Korrekt**: Alle Anforderungen zusammen sind korrekt, wenn das zu erstellende Softwaresystem sie erfüllen soll. Der Auftraggeber und/oder die Benutzer können sagen, ob die Anforderungen die Bedürfnisse reflektieren.
- **Vollständig**: Alle Anforderungen zusammen müssen alle relevanten Eigenschaften des Softwaresystems festlegen. Eine solche Forderung ist sinnvoll, wenn eine Softwareentwicklung nach dem Wasserfallmodell oder dem inkrementellen Modell erfolgt. Wird jedoch ein evolutionäres Modell oder ein agiles Modell verwendet, dann werden zu Beginn nur ein Teil der Anforderungen festgelegt (siehe »Lehrbuch für Softwaretechnik – Softwaremanagement«). Auch ist zu Beginn der Gesamtumfang des Softwaresystems noch nicht bekannt.
- **Konsistent**: Alle Anforderungen müssen untereinander widerspruchsfrei sein.
- **Abhängigkeiten angebbar**: Abhängigkeiten zwischen Anforderungen müssen sichtbar und nachverfolgbar sein, z. B. durch Angabe der Anforderungsnummern abhängiger Anforderungen.
- **Modifizierbar** *(modifiable)*: Änderungen an den Anforderungen können einfach, vollständig und konsistent unter Beibehaltung der vorhandenen Struktur und des verwendeten Stils durchgeführt werden. Die Anforderungsspezifikation muss dazu ein In-

haltsverzeichnis, einen Index und explizite Querverweise besitzen. Anforderungen dürfen nicht redundant sein, d. h. eine Anforderung darf nur einmal vorhanden sein.

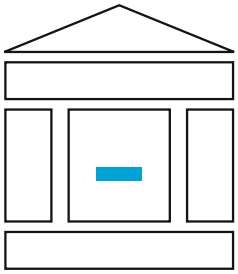
- **Erweiterbar:** Neue Anforderungen können einfach unter Beibehaltung der vorhandenen Struktur und des verwendeten Stils hinzugefügt werden.
- **Im Umfang angemessen**: Die notwendigen Anforderungen müssen in ausreichender Detaillierung und ausreichendem Präzisierungsgrad beschrieben werden. Es darf nur das *Was*, nicht das *Wie* beschrieben werden.
- **Nach verschiedenen Sichten auswertbar**: Die Anforderungsspezifikation muss nach verschiedenen Sichten – entsprechend dem Bedarf der einzelnen *Stakeholder* – sortierbar und gruppierbar sowie in verschiedenen Granularitätsstufen darstellbar sein.

Anforderungsspezifikation als Teil einer Systemspezifikation
Wird eine Anforderungsspezifikation parallel zu einer Systemspezifikation entwickelt, d. h. das zu entwickelnde Softwaresystem ist Bestandteil eines softwareintensiven Systems, dann muss darauf geachtet werden, dass die Anforderungsspezifikation kompatibel mit der Systemspezifikation ist, sonst ist sie nicht korrekt. Liegt bereits eine Systemspezifikation vor, dann muss von vornherein darauf geachtet werden, dass die Anforderungen der Systemspezifikation durch die Anforderungsspezifikation erfüllt werden.

Von der Theorie zur Praxis
In der Praxis ist man in der Regel weit davon entfernt, die obigen Qualitätskriterien für Anforderungen einzuhalten. Kunden – ob intern oder extern – werden die Wünsche an das neue Softwaresystem weder systematisch, noch strukturiert, noch vollständig beschreiben. Normalerweise wird zwischen abstrakten Angaben und konkreten Wünschen, zwischen allgemeinen Aussagen und spezifischen Festlegungen hin- und hergesprungen. Oft erfolgt eine fallorientierte Darstellung der Wünsche: »In der Situation A sollte das System dies und jenes tun«. Diese Problematik entsteht dadurch, dass die Gesprächspartner kein vollständiges Modell des zu entwickelnden Systems »im Kopf« haben.

Aufgabe des *Requirements Engineer* ist es, in Zusammenarbeit mit den *Stakeholdern* die oben aufgeführten Qualitätskriterien für Anforderungen möglichst zu erreichen.

18 Anforderungsattribute



Zu jeder Anforderung sind Attribute zu dokumentieren, die bei der Anforderungserfassung eingetragen oder im Laufe des *Requirements Engineering* ergänzt werden. Erfolgt die Anforderungsdokumentation werkzeuggestützt, dann kann auf die Angabe der Attribute hingewiesen werden bzw. können Voreinstellungen vorgenommen werden. Bei der Verwendung von Textsystemen können Anforderungsschablonen verwendet werden. Die Attribute hängen teilweise von der Anforderungsart ab.

Folgende Attribute können erfasst werden (siehe auch [Pohl07, S. 257 ff.], [Rupp07, S. 385 f.], [Eber08, S. 161 f.]):

Identifikation

- **Identifikator**: Dient zur eindeutigen Identifikation und Referenzierung, z. B. durch eine Nummer oder einen Bezeichner.
- **Kurzbezeichnung der Anforderung** (optional): Eindeutige, charakterisierende Bezeichnung, z. B. sprechender Name.

Inhalt

- **Anforderungstyp**: Gibt den Typ der Anforderung an, z. B. funktionale Anforderung.
- **Beschreibung der Anforderung**: Die Anforderung wird informal, semiformal oder formal beschrieben, z. B. in Form eines strukturierten Satzes (siehe »Natürlichsprachliche Anforderungen«, S. 481) oder eines Klassendiagramms.
- **Anforderungssicht** (bei funktionalen Anforderungen): Angabe, ob die Statik, Dynamik oder Logik des Produkts oder eine Kombination davon beschrieben wird, z. B. Angabe, dass die Anforderung die Dynamik beschreibt.
- **Querbezüge** (optional): Bezüge und Abhängigkeiten zu anderen Anforderungen, z. B. hängt ein Erfassungsformular von den Datenanforderungen ab.
- **Status des Inhalts**: Gibt an, wie vollständig die Anforderung beschrieben ist, z. B. Idee, grober Inhalt, detaillierter Inhalt.
- **Abnahmekriterien**: Angaben, wie die Erfüllung der Anforderung bei der Abnahme überprüft werden soll, z. B. durch Angabe eines Testszenarios.
- **Schlüsselwörter** (optional): Charakteristische Schlüsselwörter, die es erlauben die Anforderungen zu filtern, z. B. GUI, Persistenz.

Management

- **Priorität der Anforderung**: Festlegung der Priorität aus Auftraggeber- und Auftragnehmersicht, z. B. hohe, mittlere oder niedrige Priorität.
- **Stabilität der Anforderung**: Maß für die Wahrscheinlichkeit, dass sich die Anforderung im Entwicklungsverlauf ändert, z. B. fest, gefestigt, volatil.

H. Balzert, *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*,
DOI 10.1007/978-3-8274-2247-7_18, © Spektrum Akademischer Verlag Heidelberg 2009

- **Kritikalität der Anforderung**: Bei einer fehlerhaften Umsetzung der Anforderung können Schäden entstehen, z. B. kann bei einer hohen Kritikalität eines technischen Systems ein Fehlverhalten zum Verlust von Menschenleben führen.
- **Entwicklungsrisiko**: Gibt an, wie hoch das Entwicklungsrisiko für die Umsetzung dieser Anforderung eingeschätzt wird, z. B. hohes Risiko bzgl. der Termineinhaltung.
- **Aufwand**: Geschätzter Aufwand für die Realisierung der Anforderung, z. B. zwei Personenmonate.
- **Konflikte** (optional): Beschreibung identifizierter Konflikte bezogen auf diese Anforderung, z. B. Sicherheitsaspekte vs. Benutzungsfreundlichkeit.

Dokumentation

- **Autor**: Person, die die Anforderung beschrieben hat bzw. für die Anforderung verantwortlich ist.
- **Quelle**: Ursprung der Anforderung und Begründung, warum die Anforderung berücksichtigt wurde, z. B. Name des *Stakeholders*, Bezeichnung des Gesetzes.
- **Version und Änderungsbeschreibung**: Versionsnummer der Anforderung und Angabe welche Änderungen gegenüber der Vorgängerversion vorgenommen wurde einschl. Begründung, z. B. V02 (Detaillierung der Anforderung).
- **Bearbeitungsstatus**: Gibt an, in welchem Bearbeitungsstatus sich die Anforderung befindet, z. B. erzeugt, in Bearbeitung, der QS vorgelegt, fertig gestellt.

Kritikalität	Art des Fehlverhaltens
hoch	Fehlverhalten kann zum Verlust von Menschenleben führen.
mittel	Fehlverhalten kann die Gesundheit von Menschen gefährden oder zur Zerstörung von Sachgütern führen.
niedrig	Fehlverhalten kann zur Beschädigung von Sachgütern führen, ohne jedoch Menschen zu gefährden.
keine	Fehlverhalten gefährdet weder die Gesundheit von Menschen noch Sachgüter.

Tab. 18.0-1: Kritikalität für technische Systeme.

Attribute für Anforderungen können unternehmensspezifisch oder projektbezogen festgelegt werden. Für jedes Anforderungsattribut ist ein Attributierungsschema festzulegen, das Folgendes festlegt (siehe [Pohl07, 257 f.]):

Eindeutiger Attributnamen, Attributsemantik, Wertebereich, Wertesemantik

Beispiel

Attributname: Kritikalität der Anforderung
Attributsemantik: Maß für die Schäden, die entstehen können, wenn die Anforderung fehlerhaft umgesetzt wird.
Wertebereich: {hoch, mittel, niedrig, keine}
Wertesemantik [V-Modell XT 06]:

19 Natürlichsprachliche Anforderungen

In den meisten Fällen werden Anforderungen in natürlicher Sprache formuliert.

Frage

Überlegen Sie, welche Vor- und Nachteile die Verwendung natürlicher Sprache für die Formulierung von Anforderungen mit sich bringt.

Antwort

Natürliche Sprachen besitzen vielfältige Vorteile:

+ Die Verwendung der natürlichen Sprache ist **einfach** – vorausgesetzt alle *Stakeholder* beherrschen die jeweilige Sprache, z. B. Deutsch.
+ Eine natürliche Sprache ist **flexibel** – sie kann abstrakte und konkrete Dinge beschreiben.
+ Und sie ist für jede Anwendungsdomäne einsetzbar – sie ist **universell**.

Sie sind jedoch in mehrfacher Hinsicht **mehrdeutig**:

- **Synonyme** (z. B. Innenstadt – City) und **Homonyme** (z. B. Bank) führen zu einer **lexikalischen Mehrdeutigkeit**.
- Eine **syntaktische Mehrdeutigkeit** zeigt das folgende Beispiel: Die letzten 10 Buchungen und die Stornierungen des Kunden werden im Fenster angezeigt. Diese Anforderung kann so interpretiert werden, dass die letzten 10 Buchungen und die letzten 10 Stornierungen angezeigt werden. Eine andere Interpretation besteht darin, die letzten 10 Buchungen und alle Stornierungen anzuzeigen.
- Kann eine Anforderung in einem Kontext unterschiedlich interpretiert werden – auch wenn keine lexikalische oder syntaktische Mehrdeutigkeit vorliegt –, dann liegt eine **semantische Mehrdeutigkeit** vor, wie folgendes Beispiel zeigt: Jeder Sensor ist mit einem Service verbunden. Dies kann bedeuten: Es gibt viele Services – jeder ist mit einem anderen Sensor verbunden. Oder: Es gibt genau einen Service, mit denen alle Sensoren verbunden sind. Die Mehrdeutigkeit entsteht durch das Verhältnis der Quantoren »jeder« und »einem«.
- Eine **referentielle Mehrdeutigkeit** zeigt das folgende Beispiel: Beim Login muss zuerst das Benutzerkennzeichen und dann das Passwort eingeben werden. Ist dies nicht korrekt, schlägt die Anmeldung fehl.

H. Balzert, *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*,
DOI 10.1007/978-3-8274-2247-7_19, © Spektrum Akademischer Verlag Heidelberg 2009

- *Vage Begriffe*, wie z. B. »neben«, können von unterschiedlichen Personen verschieden interpretiert werden: Der Sensor muss neben der Tür angebracht werden.

Es gibt im Wesentlichen drei Möglichkeiten, um die Probleme der natürlichen Sprache zu reduzieren oder zu vermeiden:

- Erstellung eines Glossars
- Benutzung sprachlicher Anforderungsschablonen
- Weitgehende Vermeidung natürlichsprachlicher Anforderungen

Erstellung eines Glossars

Durch die Definition von Begriffen in Form eines Glossars werden lexikalische Mehrdeutigkeiten vermieden. Ein Glossareintrag sollte folgende Struktur besitzen:

- **Begriff**: zu definierender Begriff
- **Definition**: Text der Definition
- **Synonyme**: Begriffe mit gleicher Bedeutung
- **Plural**: Wenn ungewöhnlich, z. B. Algorithmus, Algorithmen
- **Kurzform**: Wenn vorhanden, z. B. QM für Qualitätsmanagement
- **Langform**: Wenn der Begriff in Kurzform angegeben ist, z. B. World Wide Web für WWW.
- **Übersetzung**: In eine oder mehrere Sprachen, wenn die Projektsprache oder die Benutzungsoberfläche mehrsprachig sind.

Zusätzlich kann ein Glossarbegriff Metainformationen wie Autor, Version usw. besitzen.

Ein Glossar kann ein eigenständiges Anforderungsartefakt sein oder Bestandteil eines anderen Artefakts – in der Regel Bestandteil der Anforderungsspezifikation. Damit auf das Glossar alle *Stakeholder* Zugriff haben, hat das Glossar als eigenständiges Artefakt Vorteile.

Benutzung sprachlicher Anforderungsschablonen

Um die syntaktische und semantische Mehrdeutigkeit bei natürlichsprachlichen Anforderungen zu reduzieren, haben sich in der Praxis Anforderungsschablonen bewährt. Die Abb. 19.0-1 zeigt eine Anforderungsschablone ohne Bedingungen (in Anlehnung an [Rupp07, S. 233]).

- Das **Prozesswort** (ganz rechts) beschreibt die geforderte Funktionalität durch ein Verb wie anlegen, löschen, berechnen.
- Im mittleren Teil wird angegeben, um welche Art von Systemaktivität es sich handelt:
  - Selbstständige Systemaktivität: Das System führt den Prozess selbstständig durch.

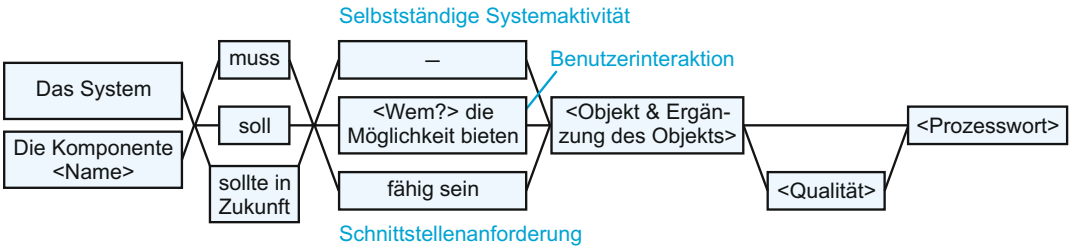


Abb. 19.0-1: Aufbau einer Anforderungsschablone ohne Bedingungen.

- Benutzerinteraktion: Das System stellt dem Benutzer die Prozessfunktionalität zur Verfügung.
- Schnittstellenanforderung: Das System führt den Prozess in Abhängigkeit von einem anderen System aus. Es ist passiv und wartet auf ein externes Ereignis.

■ Die rechtliche Verbindlichkeit einer Anforderung kann mit Hilfsverben wie folgt festgelegt werden:

- muss: rechtlich bindend
- soll (kann): dringend empfohlen
- sollte in Zukunft: wahrscheinlich ein zukünftige Anforderung

■ Die nähere oder ergänzende Bestimmung des Prozesswortes wird durch <Objekt & Ergänzung des Objekts> angegeben.

■ Nichtfunktionale Anforderungen legen in der Regel <Qualitäten> in Bezug auf das Prozesswort fest.

Beispiele: SemOrg

○ Benutzerinteraktion:
Das System muss *dem Kunden* die Möglichkeit bieten, sich über *Seminare und Veranstaltungen* zu informieren.

○ Benutzerinteraktion:
Das System soll *dem Seminarsachbearbeiter* die Möglichkeit bieten, *Seminare und Veranstaltungen* mit *selbst erstellten Suchanfragen auszuwerten.*

○ Selbstständige Systemaktivität:
Das System muss die *Kundendaten permanent speichern.*

○ Schnittstellenanforderung:
Das System muss fähig sein, dem Buchhaltungssystem *Rechnungsdatensätze mindestens einmal am Tag* zur Verfügung zu *stellen.*

Mit manchen Anforderungen sind logische und zeitliche Bedingungen verknüpft. Eine entsprechend erweiterte Schablone zeigt die Abb. 19.0-2.

Beispiel

○ Selbstständige Systemaktivität:
Falls ein Kunde im Zahlungsverzug ist, muss das System *eine neue Buchung nicht erlauben.*

✚ Wie die Beispiele zeigen, führt der Einsatz einer Schablone zu einer Vereinheitlichung der Anforderungsbeschreibungen und zu einer eindeutigeren Semantik.

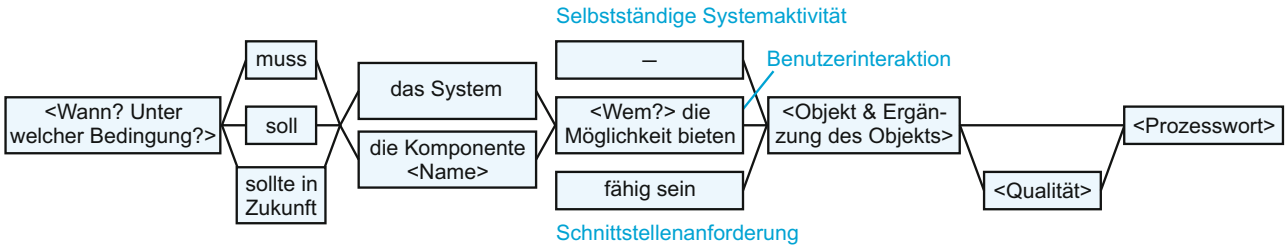


Abb. 19.0-2: Aufbau einer Anforderungsschablone mit Bedingungen.

- Auf der anderen Seite ist es schwierig, alle Anforderungen in Form solcher Sätze zu formulieren – manche Festlegungen sind schwer zu lesen.

Weitgehende Vermeidung natürlichsprachlicher Anforderungen

In der Softwaretechnik gibt es etablierte formale und semiformale Basiskonzepte (siehe »Basiskonzepte«, S. 99), die eine präzisere und eindeutigere Festlegung von Anforderungen erlauben, auch verglichen mit schablonenbasierten Anforderungsbeschreibungen. Da auf der Auftraggeberseite auch immer häufiger Informatiker tätig sind, sollte es bei Kenntnis der Basiskonzepte kein Problem sein, die Anforderungen auf der Auftraggeber- und der Auftragnehmerseite eindeutig zu interpretieren.

Nach Auffassung des Autors sollten daher auch in der Anforderungsspezifikation immer Basiskonzepte eingesetzt werden – immer, wenn dies möglich ist – und dadurch weitgehend auf natürlichsprachliche Anforderungen verzichtet werden.