11 Dokumentation von Szenarien 141
11.1 Textuelle Dokumentation narrativer Szenarien 141
11.2 Strukturierte, textuelle Dokumentation von Szenarien 142
11.3 Referenzschablone zur Dokumentation von Use Cases 147
11.4 Elf Regeln zur Formulierung natürlichsprachlicher Szenarien 151
11.5 Sequenzdiagramme 156
11.6 Aktivitätsdiagramme 159
11.7 Use-Case-Diagramme 162
11.8 Eignung von Szenarien 166

12 Verwendung von Zielen und Szenarien 167
12.1 Vorteile der Zielorientierung 167
12.2 Vorteile der Szenarioverwendung 170
12.3 Wechselwirkungen zwischen Zielen und Szenarien 174
Literaturempfehlung für Teil III.b 180

Teil III.c Lösungsorientierte Anforderungen 181

13 Traditionelle Anforderungsperspektiven 183
13.1 Ziele, Szenarien und lösungsorientierte Anforderungen 183
13.2 Die drei traditionellen Perspektiven 184
13.3 Datenmodelle 187
13.4 Funktionsmodelle 189
13.5 Verhaltensmodelle 195

14 Objektorientierte Modelle 199
14.1 Objektorientierte Modellierung 200
14.2 Dokumentation der Perspektiven in UML 201
14.3 Integration der Sichten in UML 205
Literaturempfehlung für Teil III.c 207

Teil IV Kernaktivitäten 211

Teil IV.a Dokumentation 215

15 Grundlagen der Dokumentation 217
15.1 Gegenstand der Dokumentation 217
15.2 Dokumentation vs. Spezifikation 219
15.3 Qualitätskriterien für Anforderungsartefakte 221
15.4 Abnahmekriterien 224

16 Natürlichsprachliche Dokumentation 229
16.1 Natürlichsprachliche Anforderungen 229
16.2 Anforderungsdokumente 231
16.3 Qualitätskriterien für Anforderungsdokumente 236
16.4 Vor- und Nachteile natürlichsprachlicher Dokumentation 239
16.5 Techniken zur Vermeidung von Mehrdeutigkeiten 244

17 Strukturierung natürlichsprachlicher Anforderungen 251
17.1 Referenzstrukturen für Anforderungsdokumente 251
17.2 Attributierung von Anforderungen 257
17.3 Attribute von Anforderungsartefakten 259
17.4 Schablonen und Informationsmodelle 267
17.5 Sichtenbildung bei natürlichsprachlichen Anforderungen 273

18 Grundlagen der konzeptuellen Modellierung 279
18.1 Modellbegriff 279
18.2 Eigenschaften von Modellen 281
18.3 Semiotik konzeptueller Modelle 283
18.4 Qualität konzeptueller Modelle 287
18.5 Modellierungssprachen 289
18.6 Modellbildung und Modellinterpretation 293

19 Modellbasierte und natürlichsprachliche Anforderungen 297
19.1 Anforderungsmodelle 297
19.2 Beziehungen zwischen Modellen und natürlichsprachlichen Anforderungen 299
19.3 Beziehungstypen 303
19.4 Technische Umsetzung 306
Literaturempfehlung für Teil IV.a 307

Teil IV.b Gewinnung 309

20 Grundlagen der Gewinnung 311
20.1 Motivation 311
20.2 Gegenstand der Gewinnung 312
20.3 Ziele und Szenarien in der Gewinnung 313
20.4 Teilaktivitäten der Gewinnung 314

21 Gewinnungstechniken 323
21.1 Bewertung der Techniken 323
21.2 Schablone zur Technikbeschreibung 324
21.3 Interview 325
21.4 Workshop 336
21.5 Beobachtung 346
21.6 Schriftliche Befragung 352
21.7 Perspektivenbasiertes Lesen 356

22 Assistenztechniken zur Gewinnung 363
22.1 Bewertung der Assistenztechniken 363
22.2 Brainstorming 364
22.3 Gewinnung durch Prototypen 370
22.4 Kartenabfrage 374
22.5 Mind Maps 381
22.6 Checklisten für die Gewinnung 385
Literaturempfehlung für Teil IV.b 392

Teil IV.c Übereinstimmung 393

23 Grundlagen der Übereinstimmung 395
23.1 Motivation 395
23.2 Gegenstand der Übereinstimmung 395
23.3 Ziele und Szenarien in der Übereinstimmung 397

24 Konfliktmanagement 399
24.1 Konfliktidentifikation 399
24.2 Konfliktanalyse 400
24.3 Konfliktauflösung 403
24.4 Dokumentation der Konfliktlösung 408

25 Übereinstimmungstechniken 409
25.1 Der Win-Win-Ansatz 409
25.2 Interaktionsmatrix 411
Literaturempfehlung für Teil IV.c 413

Teil V Validierung 415

26 Grundlagen der Validierung 419
26.1 Motivation und Ziele 419
26.2 Validierung vs. Verifikation 422
26.3 Validierung der Ausgaben: Drei Qualitätstore 424
26.4 Validierung der Eingaben: Kontextbetrachtung 428
26.5 Validierung der Aktivitätsdurchführung 431
26.6 Drei-Ebenen-Modell für die Validierung 432
26.7 Ziele und Szenarien in der Validierung 434
26.8 Prinzipien der Validierung 436

27 Validierungstechniken 443
27.1 Inspektionen 444
27.2 Reviews 450
27.3 Walkthroughs 453
27.4 Vergleich: Inspektionen, Reviews und Walkthroughs 455
27.5 Perspektivenbasiertes Lesen 456
27.6 Validierung durch Prototypen 459

28 Assistenztechniken für die Validierung 465
28.1 Checklisten für die Validierung 465
28.2 Konstruktion von Artefakten 474
Literaturempfehlung für Teil V 489

Teil VI Management 491

29 Drei Managementaktivitäten 495
29.1 Überblick über die drei Managementaktivitäten 495
29.2 Beobachtung des Systemkontexts 497
29.3 Management der Aktivitäten 499

30 Nachvollziehbarkeit von Anforderungen 505
30.1 Grundlagen der Nachvollziehbarkeit 505
30.2 Differenzierung der Anforderungsnachvollziehbarkeit 508
30.3 Typen von Nachvollziehbarkeitsbeziehungen 510
30.4 Repräsentation von Nachvollziehbarkeitsinformationen 515
30.5 Unternehmens-/projektspezifische Definition der Nachvollziehbarkeit 519

31 Priorisierung von Anforderungen 527
31.1 Grundlagen der Anforderungspriorisierung 527
31.2 Vorbereitungsaktivitäten für die Priorisierung 528
31.3 Techniken zur Anforderungspriorisierung 532

32 Änderungsmanagement für Anforderungen 545
32.1 Konfigurationsmanagement 545
32.2 Ursachen für Änderungen von Anforderungen 549
32.3 Das Änderungsgremium 552
32.4 Änderungsantrag 553
32.5 Vorgehen für das Änderungsmanagement 554
Literaturempfehlung für Teil VI 560

Teil VII Die ziel- und szenariobasierte Requirements-Engineering-Methode COSMOD-RE 561

33 Die COSMOD-RE-Methode 565
33.1 Hardware/Software-Codesign 566
33.2 Codesign von Anforderungen und Entwurf 567
33.3 Abstraktionsebenen 568
33.4 Codesign-Prozesse 572
33.5 Die fünf Kernaktivitäten 577
33.6 Einsatz von COMOD-RE 589

34 Anwendung von COSMOD-RE für ein Fahrerassistenzsystem 591
34.1 Fahrerassistenzsystem: Adaptive Cruise Control (ACC) 591
34.2 System-Codesign 591
34.3 Funktionsgruppen-Codesign 598
34.4 Komponenten-Codesign 602

Teil VIII Weiterführende Themen 603

35 Anforderungsbasiertes Testen 607
35.1 Motivation 607
35.2 Grundlagen des Testens 608
35.3 Die Rolle von Szenarien beim Testen 612
35.4 Anforderungsbasierte Bestimmung von Testfällen 614
35.5 Die ScenTED-Technik 617

36 Requirements Engineering und CMMI 625
36.1 Grundlagen des CMMI 625
36.2 Prozessgebiet „Anforderungsmanagement“ 628
36.3 Prozessgebiet „Anforderungsentwicklung“ 635

37 Requirements Engineering in der Produktlinienentwicklung 649
37.1 Kernkonzepte der Produktlinienentwicklung 649
37.2 Requirements Engineering für Produktlinien 657
37.3 Domänen-Requirements-Engineering 659
37.4 Applikations-Requirements-Engineering 670

38 Ziel- und szenariobasierte Werkzeugauswahl 685
38.1 Motivation für den Werkzeugeinsatz 685
38.2 Marktorientierte Verfahren 688
38.3 Vorbereitung der individuellen Werkzeugauswahl 689
38.4 Bewertungsrahmen für Requirements-Management-Werkzeuge 690
38.5 Ziel- und szenariobasierte Werkzeugauswahl 692
Literaturempfehlung für Teil VIII 696

Anhang 699

Unter Mitarbeit von 701

Glossar 703

Literatur 717

Index 737

Teil I

Grundlagen und Rahmenwerk

Klassifikation Teil I Grundlagen und Rahmenwerk	Basis	Fortgeschritten
1 Motivation		
1.1 Softwareintensive Systeme	✔	
1.2 Bedeutung des Requirements Engineering	✔	
2 Anforderungen		
2.1 Der Anforderungsbegriff	✔	
2.2 Anforderungsarten	✔	
2.3 Problem vs. Lösung	✔	
3 Entwicklung zum kontinuierlichen Requirements Engineering		
3.1 Traditionelle Systemanalyse	✔	
3.2 Essenzielle Systemanalyse		✔
3.3 Requirements Engineering als Phase	✔	
3.4 Nachteile der Systemanalyse und des phasenbezogenen Requirements Engineering	✔	
3.5 Kontinuierliches Requirements Engineering		✔
4 Das Requirements-Engineering-Rahmenwerk		
4.1 Überblick über das Rahmenwerk	✔	
4.2 Vier Kontextfacetten	✔	
4.3 Die fünf Requirements-Engineering-Aktivitäten	✔	
4.4 Die drei Arten von Anforderungsartefakten	✔	

Teil I – Grundlagen und Rahmenwerk

Überblick

Requirements Engineering ist ein entscheidender Erfolgsfaktor für Entwicklungsprojekte. Dieser Teil behandelt:

- *den Begriff der Anforderung,*
- *die Unterscheidung zwischen funktionalen Anforderungen, Qualitätsanforderungen und Rahmenbedingungen,*
- *die Entwicklung des Requirements Engineering von der Systemanalyse hin zum kontinuierlichen Requirements Engineering,*
- *den Unterschied zwischen Problemdefinition und Lösungsbeschreibung sowie*
- *unser Rahmenwerk für das Requirements Engineering, das aus folgenden Elementen besteht:*
  - Vier Kontextfacetten
  - *Drei Kernaktivitäten*
  - *Zwei Querschnittsaktivitäten*
  - *Drei Arten von Anforderungsartefakten*

Die Struktur dieses Buches wurde aus unserem Rahmenwerk für das Requirements Engineering abgeleitet und wird am Ende dieses Teils erläutert.

1 Motivation

In diesem Kapitel vermitteln wir Ihnen:

- *Die wesentlichen Herausforderungen bei der Entwicklung softwareintensiver Systeme*
- *Die Bedeutung des Requirements Engineering für die Entwicklung softwareintensiver Systeme*

1.1 Softwareintensive Systeme

Zwei Arten von softwareintensiven Systemen

In fast allen Industriebereichen, z.B. in der Automobilindustrie, in der Unterhaltungsindustrie, im Finanz- und Dienstleistungssektor, bildet Software zunehmend das Fundament für die Realisierung innovativer Funktionen und Services. Systeme, deren wesentliche Eigenschaften durch Software realisiert sind, werden als „softwarebasierte“ bzw. „softwareintensive“ Systeme bezeichnet. Wir unterscheiden zwei Arten von softwareintensiven Systemen:

Im Wesentlichen Software

- *Informationssysteme:* Ein Informationssystem erfasst, speichert, transformiert, überträgt, und/oder verarbeitet Informationen. Ziel eines Informationssystems ist es, Nutzern (Menschen oder anderen Systemen) gewünschte Informationen zur Verfügung zu stellen. Ein Beispiel für ein Informationssystem ist ein Kontoverwaltungssystem einer Bank, das Kunden die Kontostände und Umsätze mitteilt und Überweisungen ermöglicht. Informationssysteme bestehen im Wesentlichen aus Software.

Hard- und Software eng integriert

- *Eingebettete Systeme:* Im Gegensatz zu Informationssystemen ist die Software bei eingebetteten Systemen nur ein – wenn auch ein sehr wichtiger – Systembestandteil. In eingebetteten Systemen ist die Software eng mit der Hardware integriert. Daher existieren oft komplexe Wechselwirkungen zwischen Hardware und Software. Ein typisches Beispiel für ein eingebettetes System ist das Anti-Blockier-System (ABS) im Automobil, bei dem die Software im engen Zusammenspiel mit der Hardware (ABS-Steuergerät, Radsensoren, Bremshydraulik etc.) das Blockieren der Räder beim Bremsen verhindert. Das Fahrzeug bleibt dadurch auch bei einer Vollbremsung oder bei glatter Fahrbahn lenkbar.

1.1.1 Beispiel: Bedeutung von eingebetteten Systemen im Automobilbereich

Software im Automobilbereich gewinnt an Bedeutung

Der Automobilbereich ist ein prominentes Beispiel für die zunehmende Bedeutung von eingebetteten Systemen und insbesondere der wachsenden Bedeutung von Software. Dies ist durch zahlreiche Studien belegt. Beispielsweise ergab eine Befragung der Unternehmensberatung McKinsey, dass sich bis zum Jahre 2015 das Verhältnis von Mechanik und Elektronik in Automobilen um etwa 10 % zugunsten der Elektronik verändern wird (vgl. [HAWK 2004]).

Marktvolumen von Software steigt dramatisch

Das Ergebnis einer Studie der Mercer Management Consulting, der Fraunhofer Gesellschaft und Bosch aus dem Jahre 2004 zeigt, dass das Marktvolumen von Software im Automobilbereich signifikante Wachstumsraten aufweist. Die Prognose für das Marktvolumen von Technikkomponenten (Elektrik, Elektronik, Sensorik, integrierte Schaltkreise (ICs) und Software) basiert auf der Befragung von Entscheidungsträgern der Automobilhersteller und Zulieferer (vgl. [Honsig 2005]). Wie in Abbildung 1–1 dargestellt, prognostizieren die Befragten u. a. einen Zuwachs des Marktvolumens für Technologiekomponenten im Automobilsektor von 127 Milliarden Euro im Jahr 2003 auf insgesamt 316 Milliarden Euro im Jahr 2015. Der Zuwachs des Marktvolumens von Software wird von 25 Milliarden Euro im Jahr 2003 auf 133 Milliarden Euro im Jahr 2015 prognostiziert. Dies bedeutet eine Steigerung des Marktvolumens für Software um ca. 530 % und somit eine durchschnittliche Steigerung um ca. 15 % pro Jahr über 12 Jahre. Die erwarteten Zuwachsraten im Bereich Software übertreffen damit bei weitem die zu erwartenden Zuwächse in den Bereichen Elektrik, Elektronik, integrierte Schaltkreise und Sensorik.

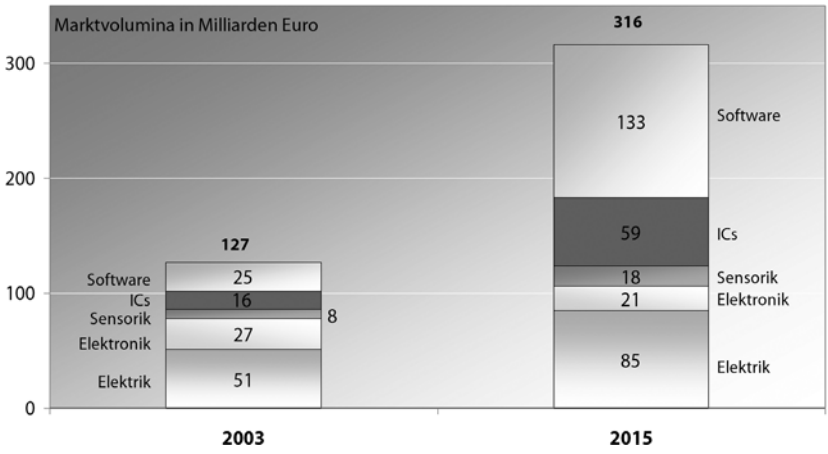


Abb. 1–1 *Vergleich des Marktvolumens von Software im Automobilbereich für das Jahr 2003 mit der Prognose für 2015 (in Anlehnung an [Honsig 2005])*

1.1.2 Herausforderungen bei der Entwicklung softwareintensiver Systeme

Die Entwicklung softwareintensiver Systeme, egal ob eingebetteter Systeme oder Informationssysteme, steht heute vor einer Reihe von Herausforderungen. Die wesentlichen Herausforderungen lassen sich wie folgt charakterisieren:

Neue Vorgehensweisen

- *Steigende Komplexität:* Die stark wachsende Anzahl von Funktionen, die durch Software realisiert werden, die zunehmende Vernetzung von Systemen, die Integration verschiedener Systeme und die Anzahl der bereitzustellenden Varianten eines Systems, z.B. um individuelle Kundenwünsche zu erfüllen, erhöhen die Komplexität von softwareintensiven Systemen. Diese Komplexität stellt die Softwareentwicklung vor neue Herausforderungen, die systematische Vorgehensweisen erfordern.

Kostendruck

- *Steigender Kostendruck:* Eine Zunahme des Verdrängungswettbewerbs und die Forderung nach niedrigen Preisen für ein Produkt führen zu stetig steigendem Kostendruck. Als Konsequenz daraus müssen softwareintensive Systeme mit geringeren Kosten entwickelt werden.

Zeitdruck

- *Kürzere Entwicklungszeit:* Zunehmender Wettbewerb führt dazu, dass Kundenwünsche schneller umgesetzt werden und innovative Produkte schneller am Markt verfügbar sein müssen. Dies bedingt, dass sich die Entwicklungszeiten für Software trotz steigender Komplexität verkürzen müssen.

Software als Innovationstreiber

- *Softwarebasierte Innovationen:* Viele, wenn nicht gar alle innovativen Systemeigenschaften werden bei softwareintensiven Systemen durch Software realisiert bzw. erst durch den Einsatz von Software ermöglicht. Software rückt daher immer mehr ins Zentrum von Produktentwicklungen und wird zunehmend zum essenziellen Erfolgsfaktor von immer mehr Produkten.

Steigende Qualität trotz Zeit- und Kostendruck

- *Steigender Qualitätsanspruch:* Die Ansprüche an die Qualität von Software, z.B. deren Verfügbarkeit, steigen ständig. Software muss daher trotz kürzerer Entwicklungszeiten und Kostenreduktion in höherer Qualität erstellt werden.

Diese Herausforderungen haben letztlich zur Konsequenz, dass innovativere, individuellere und komplexere Systeme schneller, mit höherer Qualität zu immer geringeren Preisen auf dem Markt platziert werden müssen.

Erfolgreiches Requirements Engineering ist bereits heute eine essenzielle Voraussetzung für eine erfolgreiche Systementwicklung (siehe Abschnitt 1.2). Ein kontinuierliches Requirements Engineering (siehe Abschnitt 3.5) gewinnt angesichts der skizzierten Herausforderungen an Bedeutung. Denn nur, wenn Anforderungen kontinuierlich in geeigneter Qualität gewonnen und dokumentiert werden, lassen sich diese Herausforderungen meistern.

1.2 Bedeutung des Requirements Engineering

1.2.1 Einfluss auf den Projekterfolg

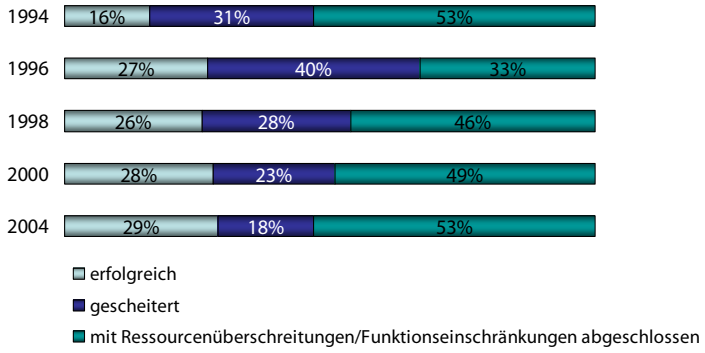
Projekterfolgsquoten

Verschiedene Studien der Standish Group nennen unzureichendes Requirements Engineering als eine der wichtigsten Ursachen für das Scheitern von Projekten. Der vielfach zitierte „Standish Group Report“ aus dem Jahre 1995 [The Standish Group 1995] berichtet, dass 52,7 % der Projekte zwar abgeschlossen wurden, allerdings das ursprünglich prognostizierte Budget um bis zu 189 % überstiegen.[1] Dabei wurden durchschnittlich nur 42% der geplanten Systemfunktionen implementiert.[2] Bei 16,1% der Projekte wurden die Zeit- und Kostenvorgaben eingehalten und sämtliche geplante Systemfunktionalitäten realisiert.[3] 31,1 % der Projekte wurden im Projektverlauf ergebnislos abgebrochen.

Vergleich der Jahre 1994–2004

Abbildung 1–2 stellt die Projekterfolgsquoten aus den „CHAOS-Studien“ der Standish Group der Jahre 1994 bis 2004 gegenüber. Die Grafik zeigt, dass der Anteil erfolgreich abgeschlossener Projekte im Jahr 2004 zwar signifikant höher ist als 1994, jedoch stagniert der Wert seit 1996 unterhalb der 30 %-Marke. In allen in Abbildung 1–2 zusammengefassten Studien liegt der Anteil der Projekte, die scheitern oder mit Ressourcenüberschreitungen bzw. Funktionseinschränkungen abgeschlossen werden, bei über 70 %. Die Situation hat sich also seit 1995 nur unwesentlich verbessert.

Abb. 1–2 *Projekterfolgsquoten aus den Standish Group (CHAOS-)Studien der Jahre 1994 – 2004 ([The Standish Group 1995; The Standish Group 1999; The Standish Group 2002; The Standish Group 2004])*



1. Bei neueren Untersuchungen aus dem Jahre 2000 liegt dieser Wert bei 45 % (vgl. [The Standish Group 2002]).
2. Diese Zahlen gelten für große Unternehmen in den Vereinigten Staaten. Bei kleineren Unternehmen waren 78,4% der Projekte erfolgreich. Im Durchschnitt wurden dabei 74,2% der geplanten Systemfunktionen realisiert (vgl. [The Standish Group 1995]).
3. Diese Zahl ist ein Durchschnittswert, der sowohl große Unternehmen als auch kleine und mittlere Unternehmen aus den Vereinigten Staaten berücksichtigt. Für große Unternehmen lag dieser Wert bei 9%.

48 % Fehlschläge aufgrund mangelndem Requirements Engineering

Bei Projekten, die nur durch Ressourcenüberschreitungen bzw. durch Einschränkung der Funktionalität abgeschlossen werden konnten, wurde hinterfragt, welche Gründe zu diesen Abweichungen geführt hatten (z.B. [The Standish Group 1995]). Abbildung 1–3 zeigt die von den Projektbeteiligten genannten Ursachen jeweils zusammen mit der Häufigkeit der Nennung. Die Gründe, die eindeutig auf Fehler im Requirements Engineering zurückzuführen sind, sind in Abbildung 1–3 durch dunkle Kreisabschnitte hervorgehoben. In der Summe liegt der Anteil der Ursachen für Projektfehlschläge, die auf Fehler oder Unzulänglichkeiten im Requirements Engineering zurückzuführen sind, bei annähernd 48 %.

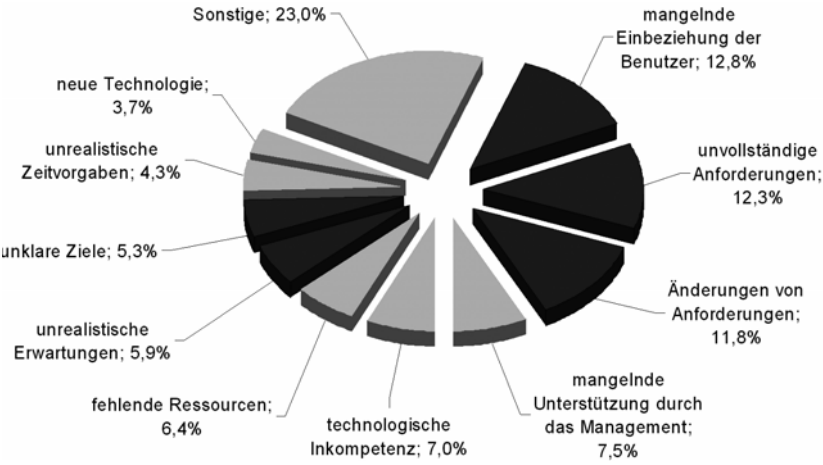


Abb. 1–3
Gründe für Ressourcenüberschreitungen und/oder Funktionseinschränkungen bei Projekten (Daten entnommen aus [The Standish Group 1995])

Bestätigung durch weitere Studien

Andere Untersuchungen kommen zu ähnlichen Ergebnissen. Eine Studie aus dem Jahr 2002, bei der zwölf Softwareunternehmen in Großbritannien befragt wurden, kam zu dem Schluss, dass Probleme mit Anforderungen die Ursache für annähernd 50 % der aufgetretenen Probleme in Entwicklungsprojekten der befragten Unternehmen waren [Hall et al. 2002]. Eine Studie des European Software Institute – der „European User Survey Analysis Report“ – stellt fest, dass die Spezifikation und das Management von Anforderungen für 50 % der in der Studie befragten Unternehmen zu den größten Herausforderungen in der Systementwicklung gehören [ESI 1996].

1.2.2 Beispiel für mangelhafte Anforderungen

Im „Forum On Risks To The Public In Computers And Related Systems“ von Peter Neumann finden sich zahlreiche Schilderungen von Fehlern, die sich bei softwareintensiven Systemen ereignet haben (*http://catless.ncl.ac.uk/Risks*). Das Spektrum reicht von Fehlern mit geringfügigen Folgen bis hin zu Fehlern mit katastrophalen Auswirkungen.

London Ambulance Service (LAS)

Viele der beschriebenen Fehlersituationen haben ihre Ursache im unzureichenden Requirements Engineering. Ein prominentes Beispiel ist das „London Ambulance Services (LAS) Computer Aided Dispatch System“ (vgl. [Finkelstein und Dowell 1996]). Durch die Entwicklung des Systems wurde beabsichtigt, die manuelle Abwicklung von Notrufen teilweise zu automatisieren. Im Falle eines eingehenden Notrufs sollte dieser von einem Mitarbeiter des LAS entgegengenommen werden. Nachdem der Ort des Notfalls von einem LAS-Mitarbeiter erfragt wurde, sollte durch das System festgestellt werden, welche Ambulanzfahrzeuge sich in der Nähe des Notfalls befinden. Ein dienstbereites und nahe am Notfallort gelegenes Ambulanzfahrzeug sollte dann zur Notfallhilfe beordert werden. In Bezug auf die Anforderungen an das „LAS Computer Aided Dispatch System“ kam es zu vielen Fehlern.

Konsequenzen mangelhafter Anforderungen

So wurden bspw. Funklöcher, in welchen die Ambulanzfahrzeuge nicht erreichbar waren, nicht berücksichtigt. Fehler in den Anforderungen führten u. a. dazu, dass die Nutzerschnittstellen an den Endgeräten der Ambulanzfahrzeuge unzureichend bzw. fehlerhaft über die Notfälle informierten. Die Fehler hatten auch zur Konsequenz, dass in kürzester Zeit die Datenbasis des Systems inkonsistent wurde. Dadurch wurden nicht dienstbereite Ambulanzfahrzeuge oder mehr Ambulanzfahrzeug als notwendig zu einem Notfallort beordert.

Weitere Beispiele

Das „LAS Computer Aided Dispatch System“ ist nur eines von vielen Beispielen für Systemversagen aufgrund eines unzureichenden Requirements Engineering. Zahlreiche Veröffentlichungen und Medienberichte befassen sich mit ähnlichen Fällen. Beispielsweise berichtet Sutcliffe von vergleichbaren Problemen beim „Eurocontrol“-Projekt [Sutcliffe 2002b] und Potts über Probleme in Projekten des amerikanischen Verteidigungsministeriums [Potts 1999].

1.2.3 Steigende Kosten für die Beseitigung mangelhafter Anforderungen

Fehler in Spezifikationen

Die Wichtigkeit des Requirements Engineering wird vielfach unterschätzt. Als Konsequenz sind Anforderungsspezifikationen oft fehlerbehaftet. Beispielsweise werden Anforderungen übersehen oder missverständlich und unvollständig spezifiziert.

Ursache für 50 % der Fehler im Quellcode

Zudem werden Mängel in der Anforderungsspezifikation häufig erst in späteren Entwicklungsphasen aufgedeckt. So sind erfahrungsgemäß etwa 50 % der im Programmquelltext gefundenen Fehler auf mangelhafte Anforderungen zurückzuführen. Gleichzeitig sind die Kosten zur Beseitigung eines Anforderungsfehlers umso höher, je später dieser Fehler im Verlaufe des Entwicklungsprojekts aufgedeckt wird [Möller 1996].

Frühzeitige Fehlerbeseitigung reduziert Kosten

Die Beseitigung eines Anforderungsfehlers, der erst beim Programmieren aufgedeckt wird, verursacht einen ca. um den Faktor 20 höheren Aufwand als eine Beseitigung des Fehlers während des Requirements Engineering. Tritt ein Anforderungsfehler erst im Abnahmetest zutage, verursacht dessen Beseitigung einen ca. um den Faktor 100 höheren Aufwand, verglichen mit der frühzeitigen Beseitigung während des Requirements Engineering (vgl. [Boehm und Basili 2001]). Es ist daher ratsam, Fehler in Anforderungen möglichst frühzeitig zu entdecken und zu beseitigen, d. h. im Requirements Engineering sicherzustellen, dass die Anforderungsspezifikation möglichst keine Fehler oder Lücken enthält.

!

Hinweis 1–1: *Steigende Bedeutung des Requirements Engineering in der Praxis*

Die wachsende Bedeutung des Requirements Engineering in der Praxis hat zahlreiche Gründe. Zu den Hauptgründen zählen:

- Das Scheitern von Projekten aufgrund von mangelhaften Anforderungsspezifikationen
- Die signifikante Kostensteigerung für die Beseitigung von Anforderungsfehlern im späteren Verlauf der Entwicklung
- Die Herausforderung, innovativere, individuellere und komplexere softwareintensive Systeme schneller, mit höherer Qualität und zu immer geringeren Preisen auf den Markt zu bringen
- Die steigende Bedeutung von softwareintensiven Systemen in zahlreichen Industriezweigen mit wachsenden Funktionsumfängen, engerem Integrationsbedarf mit anderen Systemen sowie differenzierterer Nutzung