2 Grundlegende Prinzipien des Requirements Engineering

Bei der Durchführung des Requirements Engineering sind neun Prinzipien zu berücksichtigen, die entscheidend für seinen Erfolg sind. In diesem Kapitel werden diese neun Prinzipien erläutert. Des Weiteren wird die Bedeutung des Systemkontexts und der Systemgrenze genauer vorgestellt, da diese die Basis für die Tätigkeiten des Requirements Engineer legen.

2.1 Neun grundlegende Prinzipien

Neun zentrale Prinzipien des Requirements Engineering

Die neun Prinzipien des Requirements Engineering sind laut [IREB-Lehrplan 2020]:

1. ***Wertorientierung***
   Anforderungen sind Mittel zum Zweck, kein Selbstzweck
2. ***Stakeholder***
   Im Requirements Engineering geht es darum, die Wünsche und Bedürfnisse der Stakeholder zu befriedigen
3. ***Gemeinsames Verständnis***
   Erfolgreiche Systementwicklung ist ohne eine gemeinsame Basis nicht möglich
4. ***Kontext***
   Systeme können nicht isoliert verstanden werden
5. ***Problem – Anforderung – Lösung***
   Ein unausweichlich ineinandergreifendes Tripel
6. ***Validierung***
   Nicht-validierte Anforderungen sind nutzlos
7. ***Evolution***
   Sich ändernde Anforderungen sind kein Unfall, sondern der Normalfall

8. *Innovation*
   Mehr vom Gleichen ist nicht genug
9. *Systematische und disziplinierte Arbeit*
   Wir können im Requirements Engineering nicht darauf verzichten

2.1.1 Prinzip 1: Wertorientierung – Anforderungen sind Mittel zum Zweck, kein Selbstzweck

Nur wirklich benötigte Anforderungen berücksichtigen

Die Spezifikation von Anforderungen bildet die Grundlage für die erfolgreiche Systementwicklung. Anforderungen sollten nie zum reinen Selbstzweck ermittelt und definiert werden. Es sollten nur die Anforderungen betrachtet werden, die zur Erfüllung der Stakeholder-Wünsche und -Bedürfnisse benötigt werden. Gleichzeitig muss sichergestellt werden, dass keine erforderliche Anforderung übersehen wird. Ähnliches gilt für den Detaillierungsgrad einer Anforderung: So viel Detail wie nötig, aber so wenig wie möglich, da jede Detaillierung einer Anforderung den Lösungsraum (evtl. unnötigerweise) einschränkt.

Kosten-Nutzen-Verhältnis

Die Wertorientierung für Anforderungen geht aber noch einen Schritt weiter und fordert den Nutzen einer Anforderung gegen die Kosten für deren Ermittlung, Dokumentation, Validierung, Abstimmung und Verwaltung sowie deren Realisierung abzuwägen. Der Nutzen einer Anforderung setzt sich dabei aus dem Beitrag der Anforderung zur Erfüllung der Wünsche und der Bedürfnisse der Stakeholder sowie dem Beitrag zur Reduktion des Risikos von Fehlschlägen und kostspieligen Nacharbeiten bei der Entwicklung des Systems zusammen.

Nicht zu viele, aber auch nicht zu wenige Anforderungen

Die Betrachtung der Kosten und des Nutzens (und damit verbunden auch die Risikobewertung) einer Anforderung ist ein gutes Hilfsmittel, um zu entscheiden, auf welche Anforderungen sich der Requirements Engineer zu einem bestimmten Zeitpunkt konzentrieren sollte. Es unterstützt das »Nicht zu viel, aber auch nicht zu wenig«-Ziel bezüglich der Definition von Anforderungen und erlaubt, die Ressourcen im Requirements Engineering zielgerichtet einzuplanen.

Definition 2–1: *Wert einer Anforderung*

Der Wert einer Anforderung ist gleich ihrem Nutzen abzüglich der Kosten für das Ermitteln, Dokumentieren, Validieren und Verwalten der Anforderung.

[IREB-Lehrplan 2020]

Die Kosten-Nutzen-Betrachtung eignet sich insbesondere bei agilen Vorgehensweisen. Eine detaillierte Kosten-Nutzen-Betrachtung ist – falls überhaupt möglich – meist sehr aufwendig. Oft reicht daher eine grobe Kosten-Nutzen-Abschätzung aus, um zu priorisieren, auf welche Anforderungen man sich aktuell konzentriert.

Insbesondere bei agiler Vorgehensweise geeignet

Kernfakten 2–1: *Wertorientierung*

Exkurs: Anforderungen sind kein Selbstzweck

Sowohl die Kosten für das Requirements Engineering als auch die Kosten für die Umsetzung und den Test etc. einer Anforderung können sich sehr stark unterscheiden. Gleiches gilt für den Nutzen einer Anforderung. Die Kosten-Nutzen-Abschätzung ist ein sehr gutes Hilfsmittel, um immer wiederkehrend zu entscheiden, auf welche Anforderungen man sich als Nächstes fokussieren sollte und welche Anforderungen zu dem jetzigen Zeitpunkt keine große Rolle spielen.

Das Kosten-Nutzen-Verhältnis einer Anforderung kann sich während der Laufzeit eines Projekts ändern. Gerade bei iterativ-inkrementellem Vorgehen sollten Sie deswegen vor der Bearbeitung einer Menge von Anforderungen immer planen, wie viel Aufwand Sie in diese Anforderungen investieren möchten.

Die Berücksichtigung der Kosten und des Nutzens einer Anforderung erfolgt in der Praxis typischerweise eher auf höheren Abstraktionsebenen, d.h. nicht auf der Ebene von detaillierten Einzelanforderungen, sondern beispielsweise für User Stories, Use Cases oder System-Features.

Insbesondere bei agilen Entwicklungsprojekten hat sich die Kosten-Nutzen-Betrachtung bewährt. Hier wird typischerweise der Return on Investment (Kosten-Nutzen-Abschätzung, ROI) je User Story bewertet. Basierend auf dem ROI wird dann die Auswahl der aktuell zu bearbeitenden User Stories priorisiert. Für wasserfallähnliche Prozesse ist die Wertorientierung nicht so gut verstanden bzw. in der Praxis wenig etabliert, auch weil eine Bewertung aller Anforderungen zu Beginn eines Projekts schwerfallen und fehlerbehaftet sein kann.

→

Beachten Sie folgende Tipps:

- Führen Sie die Kosten-Nutzen-Betrachtungen für Anforderungen auf höheren Abstraktionsebenen durch.
- Wiederholen Sie Kosten-Nutzen-Betrachtungen von Anforderungen zu einem späteren Zeitpunkt (in einer späteren Iteration).
- Fokussieren Sie zuerst auf Anforderungen mit einem guten Kosten-Nutzen-Verhältnis.
- Auch wenn Kosten und Nutzen nur schwer abschätzbar sind, gilt hier: besser grob als gar nicht.
- Halten Sie den Aufwand für eine Kosten-Nutzen-Abschätzung gering – besser eine grobe Abschätzung als eine detaillierte, kostenaufwendige Analyse.
- Fragen Sie sich bei der Erhebung und Dokumentation von Anforderungen regelmäßig, ob
  - die Anforderung wirklich benötigt wird und
  - der Aufwand für die Bearbeitung der nächsten Ebene feingranularer Anforderungen gerechtfertigt ist.

2.1.2 Prinzip 2: Stakeholder – Im Requirements Engineering geht es darum, die Wünsche und Bedürfnisse der Stakeholder zu befriedigen

Identifikation und Einbeziehung der richtigen Stakeholder

Für ein erfolgreiches Requirements Engineering ist es wichtig, die richtigen Personen (Stakeholder) zum richtigen Zeitpunkt einzubeziehen. Die Stakeholder geben vor, was das zu entwickelnde System tun soll, wie es auszusehen hat, kurz gesagt, welche Anforderungen es erfüllen soll. Die richtigen Stakeholder zu identifizieren, ihre Wünsche und Bedürfnisse zu ermitteln und zu verstehen und sie bei Bedarf zu konsolidieren ist eine der zentralen Aufgaben eines Requirements Engineer. Durch die richtige Einbeziehung der Stakeholder wird das Risiko minimiert, ein System zu entwickeln, das von den Stakeholdern nach Auslieferung nicht akzeptiert wird.

Typische Rollen

Nach [IREB-Lehrplan 2020] sind typische Beispiele für Stakeholder-Rollen:

- Benutzer, der das zu entwickelnde System bedienen wird
- Kunde, dem das fertige System geliefert wird
- Auftraggeber, der für die Entwicklung und das auszuliefernde System zahlt
- Betreiber, der das System im Einsatz administriert
- Regulierungsbehörde, die einzuhaltende Normen, Standards und Gesetze verantwortet

Darüber hinaus können zahlreiche weitere Rollen oder Gruppen von Stakeholdern für Ihr Vorhaben relevant sein, wie beispielsweise:

- Projektleitung/-management
- Produktion
- Produktmanagement, Vertrieb
- Einkauf
- Qualitäts-/Prozessabteilung, Security-/Safety-Beauftragte
- Domänenexperte
- Architekt (des betrachteten Systems oder des einzubettenden Systems)

Berücksichtigung von Nutzern existierender Systeme

Ersetzt das zu entwickelnde System ein bestehendes System oder stellt es eine Weiterentwicklung eines bestehenden Systems dar, so sind die aktuellen Systemnutzer auch als wichtige Stakeholder-Rollen einzubeziehen. Sie können wichtiges Feedback zum bestehenden System (z.B. Verbesserungsmöglichkeiten, auftretende Fehler, fehlende Funktionen) geben, die schon in den Anforderungen beachtet werden müssen.

Ein Stakeholder, mehrere Rollen

Für die Besetzung der o.g. Stakeholder-Rollen gilt, dass eine Person durchaus auch mehrere Rollen einnehmen kann. Für Stakeholder-Rollen mit einer hohen Anzahl an Einzelpersonen oder falls kein Vertreter für eine Rolle existiert oder ansprechbar ist, können fiktive, archetypische Beschreibungen, sogenannte Personas, als Ersatz verwendet werden (siehe Abschnitt 4.1.1).

Konflikte und Widersprüche

Stakeholder haben oft unterschiedliche Wünsche, Bedürfnisse und Ansichten bezüglich des zu entwickelnden Systems, die zu konträren (gegensätzlichen, konkurrierenden) Anforderungen führen können. Eine Aufgabe des Requirements Engineering ist es, diese Konflikte zu identifizieren und sie mit den Stakeholdern zusammen zu lösen (siehe Abschnitt 4.3), um zu einer widerspruchsfreien Menge von Anforderungen zu kommen.

Kernfakten 2–2: *Stakeholder*

Exkurs: Wichtige Anforderungsquellen zusätzlich zu Stakeholdern

Bitte beachten Sie: Das hier angesprochene Prinzip der Einbeziehung der richtigen Stakeholder darf in der Praxis nicht nur auf die Stakeholder bezogen werden, da ein System zwar primär, aber nicht nur die Wünsche der Stakeholder erfüllen muss. In der Regel existieren weitere wichtige Quellen für Anforderungen an das System wie beispielsweise Dokumente sowie existierende oder konkurrierende Systeme. Eine detaillierte Betrachtung der wichtigen Anforderungsquellen finden Sie in Abschnitt 4.1. Diese Quellen werden nur teilweise (oder gar nicht) von einem Stakeholder verantwortet, mit dem Sie in persönlichen Kontakt treten können.

2.1.3 Prinzip 3: Gemeinsames Verständnis – Erfolgreiche Systementwicklung ist ohne eine gemeinsame Basis nicht möglich

Anforderungen bergen die Gefahr, dass sie von Stakeholdern unterschiedlich verstanden werden. Deswegen muss ein gutes Requirements Engineering ein gemeinsames Verständnis aller Beteiligten (Stakeholder, Entwickler, Requirements Engineer etc.) sicherstellen.

Wir unterscheiden zwischen zwei Arten des gemeinsamen Verständnisses: explizites gemeinsames Verständnis und implizites gemeinsames Verständnis.

Dokumentiertes Verständnis

Ein ***explizites gemeinsames Verständnis*** wird während des Requirements Engineering durch Dokumentation und Reflexion bzw. Abnahme der Anforderungen erreicht. Explizites gemeinsames Verständnis wird beispielsweise durch Glossare, erstellte und validierte Prototypen, erfolgte Kosten-Nutzen-Abschätzungen oder durch Entscheidungen dokumentiert.

Insbesondere bei agilen Vorgehensweisen wichtig

Ein ***implizites gemeinsames Verständnis*** ist in allen Bereichen wichtig, in denen kein explizites gemeinsames Verständnis dokumentiert wird bzw. dokumentiert werden kann. Dies ist insbesondere bei agilen Vorgehensweisen der Fall, bei denen beispielsweise die Anforderungen an das System meist nicht vollständig dokumentiert werden. Ein implizites gemeinsames Verständnis basiert auf dem gemeinsamen Wissen aller Beteiligten. Der Aufbau des impliziten gemeinsamen Verständnisses wird durch intensiven Austausch und enge Zusammenarbeit untereinander aufgebaut bzw. ausgebaut.

Beide Arten von Verständnis können durch verschiedene Techniken und Ansätze unterstützt werden. So hilft eine intensive Zusammenarbeit, Kommunikation und Reflektion über Arbeitsergebnisse, um das Verständnis zu erhöhen oder um zumindest zu erkennen, wo noch De-

fizite behoben werden müssen. Dies wird insbesondere durch Vorgehensweisen mit kurzen Feedbackzyklen unterstützt (siehe Abschnitt 5.2).

Das gemeinsame Verständnis über die Anforderungen wird durch die Verwendung von zusätzlichen Arbeitsprodukten unterstützt. So können die Anforderungen durch Dokumentation der benutzten Begrifflichkeiten in einem Glossar (Abschnitt 3.5) oder durch prototypische Implementierungen (Abschnitt 3.7) angereichert werden. Weiterhin kann als Basis für ein gemeinsames Verständnis ein bestehendes, den Beteiligten bekanntes System als Bezugspunkt herangezogen werden.

Der Requirements Engineer kann sich bezüglich eines gemeinsamen Verständnisses über die Anforderungen relativ sicher sein, wenn die Beteiligten z.B. bei einer Kosten-Nutzen-Abschätzung, bei der Untersuchung von Prototypen oder bei dem Erstellen von Beispielen für erwartete Ergebnisse zu ähnlichen Ergebnissen kommen.

Grundsätzlich fördert ein ähnliches Wissen der Beteiligten über die Anwendungsdomäne das gemeinsame Verständnis. Hilfreich ist auch, wenn ein Vertrauen zwischen den Beteiligten, vielleicht entstanden in einer früheren Zusammenarbeit, existiert. Zudem erleichtert eine gemeinsame Wertvorstellung der Beteiligten (und damit implizit eine gleiche/ähnliche Kultur) das gemeinsame Verständnis und reduziert potenzielle Konflikte (siehe Abschnitt 4.3).

Kernfakten 2–3: *Gemeinsames Verständnis*

Exkurs: Gemeinsames Verständnis schaffen

In der Praxis ist es nicht möglich, das gesamte gemeinsame Wissen zu dokumentieren. Es gibt daher immer zusätzlich zu dem dokumentierten, gemeinsamen Wissen auch implizites gemeinsames Wissen. Darüber hinaus kann auch dokumentiertes Wissen durchaus missverstanden werden. Im Requirements Engineering wird daher sehr viel implizites gemeinsames Wissen vorausgesetzt.

Die Kunst im Requirements Engineering liegt darin, die richtige Mischung aus implizitem und explizitem Wissen hinzubekommen, d.h. nur so viel gemeinsames Wissen wie nötig zu explizieren und somit die hiermit verbundenen Aufwände bzw. Kosten im Rahmen zu halten.

→

Expliziertes Wissen lässt sich leicht teilen, diskutieren, prüfen und, bei gleichem Verständnis, auch umsetzen. Das Explizieren kostet aber auch Zeit und Geld. Je formaler die Art der Dokumentation, desto weniger Missverständnisse sind möglich (vorausgesetzt alle Beteiligten verstehen die Formalisierungen und sind bereit, die Formalisierungsschritte mitzutragen). Formalisierung von Anforderungen wird u.a. durch Templates für die natürliche Sprache und den Einsatz von semiformalen Beschreibungssprachen (UML, BPMN, SysML, …) unterstützt (siehe Kapitel 3). Diese Formalisierung birgt allerdings auch einige Nachteile. Beispiele für Nachteile sind die mit der Formalisierung verbundenen Kosten für die Erstellung und die durchaus, im Vergleich zur natürlichen Sprache, schlechtere Verständlichkeit für einen Teil der Stakeholder.

Prüfen Sie anfangs, wie stabil und weitreichend ihr implizites gemeinsames Verständnis ist, und erweitern Sie es durch den Einsatz verschiedener Methoden und organisatorischer Maßnahmen, wie z.B.

- dem gemeinsamen Erarbeiten von Artefakten (wie Anforderungen, User Stories, Use Cases, grafischen Modellen, ….);
- Storytelling, so vermitteln z.B. Vision Stories allen Beteiligten auf einprägsame Weise die Vision für die Systementwicklung;
- dem Einsatz von Beobachtungstechniken, um einen guten gemeinsamen Eindruck der Einsatzumgebung zu erhalten.
- Setzen Sie auf »osmotische Kommunikation« im Team – Menschen, die in einem Raum sitzen, bekommen automatisch sehr viele Informationen von den anderen im Raum sitzenden Personen mit.
- Kurze Wege befördern Kommunikation – vermeiden Sie räumliche Trennungen Ihrer Teams.
- Versuchen Sie die Besetzung von Projektteams stabil zu halten. Fluktuation und häufiger Personalwechsel erfordern jedes Mal einen Wissenstransfer. Insbesondere für das implizite gemeinsame Wissen ist dies schwierig, fehlerbehaftet und aufwendig.
- Arbeiten Sie mit kleinen Teams bzw. Subteams. Zu große Gruppen behindern eine gute Kommunikation.
- Arbeiten Sie in kurzen Zyklen – nur so können Sie prüfen, ob Ihr implizites gemeinsames Verständnis vorhanden ist.

→

Prüfen Sie regelmäßig, wie deckungsgleich Ihr Verständnis ist – egal, ob Sie das Wissen inzwischen expliziert haben oder es nur implizit geteilt wird, indem Sie z. B.

- Beispiele oder Abnahmekriterien oder Testfälle ausarbeiten und diskutieren;
- User Stories, Use Cases oder Anforderungen bezüglich ihres Entwicklungsaufwands abschätzen;
- Prototypen erstellen oder bezüglich ihrer Tauglichkeit bewerten;
- Anforderungen gemeinsam validieren;
- Anforderungen bewerten/priorisieren/ihre Auswirkung auf die Architektur bewerten;
- über relevante Stakeholder diskutieren;
- den Nutzen oder Return on Investment von Anforderungen besprechen.

Die hier vorgeschlagenen Methoden und Techniken setzen auf eine gute und intensive Zusammenarbeit der Projektbeteiligten. Es können natürlich weitere Techniken im Projekt eingesetzt werden, die die gemeinsame inhaltliche Diskussion über die Arbeitsprodukte des Requirements Engineering fördern.

2.1.4 Prinzip 4: Kontext – Systeme können nicht isoliert verstanden werden

Umgebung/Kontext von Systemen

Ein System wird nach seiner Fertigstellung immer in einer bestimmten Umgebung oder auch mehreren Umgebungen eingesetzt. Unter System verstehen wir hier soziotechnische Systeme, die sowohl technische, organisatorische als auch Nutzungsaspekte umfassen. Sei es, dass das System Geschäftsprozesse in einer Organisation unterstützen soll oder dass es als Teil eines übergeordneten Systems gewisse Aufgaben übernimmt. Damit sind die Anforderungen an das geplante System von seiner Umgebung abhängig. Beispielsweise beeinflussen die geplante Systemnutzung und die Eigenschaften der Nutzer die Anforderungen an das System. Aber auch andere Aspekte müssen berücksichtigt werden, wie andere Systeme, mit denen das System im operativen Einsatz zusammenarbeiten soll, Geschäftsprozesse, die es unterstützen soll, oder die Organisationen, in denen das System eingesetzt werden soll.

Anforderungen existieren nur bezüglich eines bestimmten Kontexts.

Generell gilt: Anforderungen an Systeme werden immer in Bezug zu einem Kontext definiert.

Systemkontext bestimmen

Zentrale Aufgabe des Requirements Engineering

Für das Requirements Engineering ist es also wichtig, den Teil der Umgebung des Systems zu identifizieren, von dem die Anforderungen an das System abhängen (vgl. [Pohl 2010]). Dieser für das System relevante Teil der Umgebung wird als Systemkontext bezeichnet.

> **Definition 2–2:** *Systemkontext*
>
> Der Systemkontext ist der Teil der Umgebung eines Systems, der für die Definition und das Verständnis der Anforderungen des zu entwickelnden Systems relevant ist.
>
> Übersetzt aus [IREB-Glossar 2020]

Annahmen über den Kontext

Bitte beachten Sie, dass der Systemkontext nicht nur für die Definition der Systemanforderungen essenziell ist (ohne Kontext keine Anforderung), sondern dass auch bestimmte Annahmen über den Systemkontext (bspw. über andere Systeme, Nutzer, physikalische Voraussetzungen, Geschäftsprozesse) gelten müssen, damit das System seine geplanten Funktionalitäten und Qualitäten erfüllen kann. Solche Annahmen über den Kontext des Systems werden auch als Domänenannahmen bezeichnet. Ist beispielsweise die Prüfung der Kreditwürdigkeit eines Kunden durch ein existierendes System (z.B. Schufa) vorgesehen, kann das zu entwickelnde neue System die Kreditwürdigkeit nur dann prüfen, wenn in seinem Kontext ein System existiert, das die benötigten Kreditwürdigkeitsdaten liefert. Ein weiteres Beispiel für eine solche Annahme ist die Einhaltung der Umgebungstemperatur, für die das System entwickelt wird (z.B. -10 bis +40 Grad). Weicht die Temperatur in der Umgebung hiervon ab, ist nicht sichergestellt, dass das System auch funktioniert.

Kontextänderungen berücksichtigen

Des Weiteren ist die Festlegung des Systemkontexts relevant, um Änderungen im Kontext im Laufe der Entwicklung zu identifizieren und die Auswirkungen der Änderungen auf die Systemanforderungen berücksichtigen zu können. Mehr zur Definition des Systemkontexts finden Sie in Abschnitt 2.2.

Im Folgenden wird die Abgrenzung von System und Kontext näher definiert. Hierfür grenzt Abbildung 2–1 die wichtigsten verwendeten Begriffe voneinander ab. System und Systemkontext werden von der Systemgrenze separiert. Der Systemkontext beschreibt den relevanten Teil der Umgebung, der mit der Kontextgrenze vom irrelevanten Teil der Umgebung abgegrenzt wird.

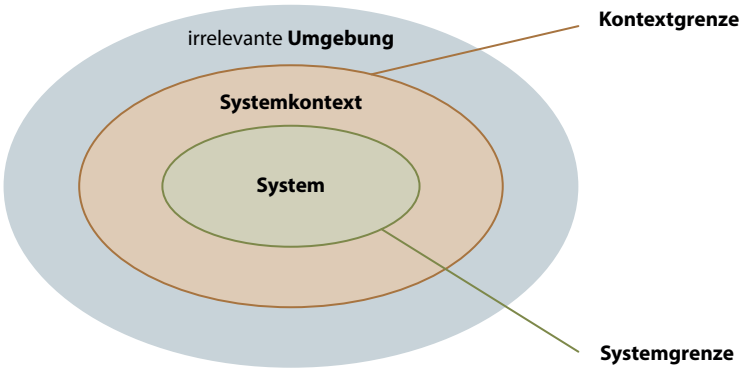


Abb. 2–1
Systemgrenze und Kontextgrenze

Bestimmen der Systemgrenze

Entscheidend für die korrekte Definition von Anforderungen an das System ist die Definition der Systemgrenze, d.h. der Abgrenzung des Systems von dessen Kontext. Mit dieser Abgrenzung entscheiden die Stakeholder, was bei der Systementwicklung verändert werden kann (Dinge innerhalb der Systemgrenze) und was nicht verändert werden darf (Dinge außerhalb der Systemgrenze, d.h. dem Kontext zugeordnete Dinge).

Definition 2–3: *Systemgrenze*

Die Systemgrenze separiert das geplante System von seinem Kontext.

Übersetzt aus [IREB-Glossar 2020]

Was kann gestaltet werden; was ist unveränderbar?

Das zu entwickelnde System wird also durch die Systemgrenze vom Kontext (Umgebung) abgegrenzt (siehe auch Abschnitt 2.2.2). Durch diese Abgrenzung werden materielle und immaterielle Aspekte (Objekte, Personen, Sachverhalte, Prozesse etc.) entweder dem System oder dem Systemkontext zugeordnet. Durch diese Zuordnung wird entschieden, ob ein materieller oder immaterieller Aspekt während der Systementwicklung verändert bzw. angepasst werden kann oder nicht.

Was darf verändert werden?

Wird ein Aspekt dem System zugeordnet (d.h., er liegt innerhalb der Systemgrenze), so kann er während der Entwicklung verändert und angepasst werden, wird er dem Kontext zugeordnet, dann darf er nicht verändert werden. Werden beispielsweise ein Ausgabegerät (z.B. ein Drucker), ein existierendes anderes System oder ein Geschäftsprozess (oder Teile davon) dem System zugeordnet – und liegen sie damit innerhalb der Systemgrenze –, dürfen diese Aspekte während der Entwicklung verändert oder ausgetauscht werden bzw. können sogar ganz

entfallen. Diese Aspekte gehören daher zum Gestaltungsspielraum für die Systementwicklung. Sie sind Gegenstand der Entwicklung und gehören somit zum »Umfang« des Systems. Der Drucker würde in diesem Falle beispielsweise frei gestaltbar sein und wäre bei einer Beauftragung auch Gegenstand der Lieferung des Auftragnehmers oder könnte auch, falls nicht durch andere Anforderungen eingeschränkt, sogar ganz wegfallen und beispielsweise durch eine elektronische Ausgabe der Dokumente ersetzt werden.

Nicht veränderbare Objekte

Werden die Aspekte im Gegensatz dazu dem Kontext des Systems zugeordnet (d.h., sie liegen außerhalb der Systemgrenze), dann dürfen sie während der Entwicklung nicht verändert bzw. angepasst werden. In diesem Falle wäre beispielsweise die Schnittstelle des vorhandenen Druckers zu berücksichtigen oder die Anforderungen des dem Kontext zugeordneten Geschäftsprozesses. Wie für andere Kontextaspekte auch, würden sich dann aus diesen Kontextobjekten wichtige Anforderungen an das System ableiten.

Richtige Abgrenzung ist essenziell.

Die korrekte Definition der Systemgrenze ist daher eine zentrale und wichtige Aufgabe des Requirements Engineering. Details zur Definition der Systemgrenze und den hierbei oft entstehenden »Grauzonen« finden Sie in Abschnitt 2.2.2.

Aufgrund der zentralen Bedeutung einer korrekten Systemabgrenzung ist es wichtig, dass der Requirements Engineer überprüft, ob das zu realisierende System vielleicht doch nur einen Teil der Anforderungen erfüllen soll, d.h. nur solche Anforderungen definiert sind, die auch tatsächlich vom System umgesetzt werden sollen. Hierfür ist es sehr wichtig, dass der Requirements Engineer für jede Anforderung prüft:

a) ob sich die gesamte Anforderung auf das zu entwickelnde System bezieht (z.B. Prüfung der Kreditwürdigkeit) oder ob Teile der Anforderung durch vorgegebene Kontextaspekte erfüllt werden (z.B. durch die Einbeziehung der Schufa bei der Prüfung der Kreditwürdigkeit);
b) ob die Anforderung unnötig durch die aktuelle Systemabgrenzung eingeschränkt wird, d.h., ob Dinge dem Kontext zugeordnet sind, die eigentlich gestaltbar sein sollten.

Systemgrenze und Umfang bzw. Scope des Systems

Systemumfang/Scope

Die Systemgrenze grenzt nicht nur das System vom Kontext des Systems ab, sondern legt mit dieser Abgrenzung auch den Umfang des zu entwickelnden Systems fest. Dabei sollte beachtet werden, dass der Begriff »System« auch organisatorische Aspekte, wie Geschäftsprozesse, miteinschließt. Laut dem IREB-Lehrplan 3.0 umfasst der Umfang die gestaltbaren Dinge (Aspekte) während der Entwicklung des Systems.

Definition 2–4: *Umfang/Scope (einer Systementwicklung)*

Der Umfang (Scope) einer Systementwicklung beinhaltet alle Dinge, die während der Entwicklung eines Systems gestaltbar sind.

Übersetzt aus [IREB-Glossar 2020]

Umgang mit dem Scope

Prinzipiell haben Sie zwei Möglichkeiten, mit dem Umfang (Scope) der Systementwicklung (d.h. die Menge der gestaltbaren Dinge) umzugehen:

a) Sie definieren zusätzlich zu der Systemgrenze noch eine Grenze für den Umfang der Systementwicklung (Scope-Grenze) und verwalten diese geeignet.
b) Sie nutzen die Systemgrenze als Festlegung des Systemumfangs, d.h., die Systemgrenze definiert den Systemumfang (Scope des Systems); die »Umfangsgrenze« bzw. »Scope-Grenze« entfällt.

Hinweis

Da die zusätzliche Berücksichtigung einer Umfangsgrenze zu signifikanten Problemen führt, raten wir Ihnen von der Definition einer Umfangs- bzw. Scope-Grenze ab! Verwenden Sie stattdessen die Systemgrenze auch als Abgrenzung für den Umfang der Systementwicklung, d.h. für die Teile, die veränderbar bzw. gestaltbar sind (siehe auch Exkurs »Probleme mit getrennter Umfang-(Scope-) Abgrenzung« auf S. 31).

Systemgrenze ändert sich

Veränderung der Systemgrenze ist der Normalfall.

Die Abgrenzung zwischen System und Kontext ist zu Beginn des Requirements Engineering typischerweise nur rudimentär verstanden und verändert sich während des Requirements Engineering. Beispielsweise kann entschieden werden, dass bestimmte Kontextaspekte jetzt doch verändert werden können, oder Aspekte, die bisher innerhalb der Systemgrenze lagen, nicht mehr verändert werden dürfen und daher dem Kontext zugeordnet werden. Oder bei einer detaillierteren Betrachtung bzw. Verfeinerung der Anforderungen kann entschieden werden, dass

Teile der Anforderung nicht mehr vom System realisiert werden sollen und stattdessen von Kontextaspekten genutzt werden. In beiden Fällen ändern sich die Anforderungen an das System!

Kontinuierliche Überprüfung der Systemgrenze ist unabdingbar.

Auch hierdurch ändert sich die Systemgrenze und somit der Umfang der Systementwicklung. Der Requirements Engineer muss diese Verschiebung erkennen, da sich hierdurch Anforderungen und Kosten für Ihr Projekt ändern werden. Die kontinuierliche Überprüfung und Anpassung der Systemgrenze ist daher sehr wichtig und eine Kernaufgabe des Requirements Engineering (siehe auch Abschnitt 2.2.2). Prüfen Sie bei jeder Anpassung die Auswirkungen der Anpassung auf die Anforderungen an das System.

Kontextgrenze

Die Kontextgrenze grenzt die für die Systementwicklung relevanten Kontextaspekte von den Aspekten in der realen Welt ab, die für die Systementwicklung nicht beachtet werden müssen, d.h. als irrelevant eingestuft werden. Liegt ein materieller oder immaterieller Aspekt innerhalb der Kontextgrenze, dann ist er im Requirements Engineering und während der Systementwicklung zu beachten. Liegt er außerhalb der Kontextgrenze, dann ist er für die Entwicklung des Systems nicht relevant, d.h., er muss während der Entwicklung des Systems nicht berücksichtigt werden.

Kontextobjekte

Eine solche Abgrenzung erfolgt beispielsweise für Personen, die befragt bzw. nicht befragt werden müssen, Annahmen über den Kontext, die beachtet oder nicht beachtet werden müssen, aber auch für existierende Systeme, relevante Gesetze, Standards und Vorschriften. Eine auch nur annähernde Zuordnung aller Aspekte der realen Welt zum relevanten Kontext oder der irrelevanten Umgebung ist selbstverständlich nicht möglich (siehe auch Abschnitt 2.2.2). Während möglichst alle im Requirements Engineering und während der Systementwicklung zu berücksichtigenden Kontextaspekte dem Systemkontext zugeordnet werden sollen (um sicherzustellen, dass kein wichtiger Aspekt übersehen wird), erfolgt eine explizite Zuordnung zur nicht relevanten Umgebung nur für Aspekte, für die explizit entschieden wurde, dass sie nicht berücksichtigt werden müssen. Beispielsweise könnte entschieden werden, dass ein bestimmter Geschäftsprozess (obwohl er evtl. relevant für die Systementwicklung sein könnte) bei der aktuellen Entwicklung nicht zu berücksichtigen ist. Um dies zu dokumentieren, wird dieser Geschäftsprozess dem nicht relevanten Kontext zugeordnet, d.h., er liegt außerhalb der Kontextgrenze.

Kernfakten 2–4: *Kontext*

Exkurs: Probleme mit getrennter Umfang-(Scope-)Abgrenzung

Im Handbuch des IREB zum CPRE Foundation Level 3.0 wird, neben der System- und Kontextgrenze, der Umfang (Scope) für eine Systementwicklung als zusätzliche Abgrenzung eingeführt. Die Scope-Grenze liegt innerhalb der Kontextgrenze und legt fest, was in der Entwicklung verändert werden darf und was nicht (siehe [IREB-Handbuch 2020], Principle 4, Seite 18/19).

Diese zusätzliche Abgrenzung ist nach unserer Auffassung nicht erforderlich. Die für die getrennte Betrachtung des Umfangs (Scopes) genannte Notwendigkeit (bspw. um in Domänenanforderungen ausgedrückte Domänenphänomene oder Domänenannahmen entsprechend berücksichtigen zu können) kann durch eine richtige Zuordnung der Aspekte zum Kontext bzw. zum System (auf der entsprechenden Betrachtungsebene wie bspw. Gesamtsystem, Subsystem etc.) adäquat adressiert werden.

Wird zusätzlich zu der Systemabgrenzung eine Abgrenzung des Umfangs (Scope) des Systems definiert, um festzulegen, welche Aspekte bei der Systementwicklung gestaltet und verändert werden dürfen, so erhöht sich hierdurch unnötig die Komplexität. Es müssen zwei Arten von Abgrenzungen durchgeführt, aktuell gehalten und verwaltet werden.

Wird ein Aspekt bezüglich der beiden Grenzen gleich eingeordnet, d. h., liegt er innerhalb der Nutzungsgrenze und der Systemgrenze oder liegt er außerhalb der Nutzungsgrenze und außerhalb der Systemgrenze, so entstehen keine zusätzlichen Probleme.

Zusätzliche Probleme entstehen, wenn Aspekte bezüglich der Systemgrenze und der Umfangsgrenze unterschiedlich zugeordnet werden.

Liegt ein Aspekt innerhalb der Umfangsgrenze (d. h., er darf verändert werden) und gleichzeitig nicht innerhalb der Systemgrenze (d. h. im Kontext des Systems), dann ist dieser Aspekt Gegenstand der Kontextbetrachtung im Requirements Engineering. Es werden daher höchstwahrscheinlich Anforderungen bezüglich dieses Aspekts erhoben und dokumentiert, da angenommen wird, dass der Aspekt nicht verändert werden darf. Die Wahrscheinlichkeit, unnötige An-

→

forderungen zu definieren, ist daher groß. Solche Anforderungen schränken u.a. die Systementwicklung unnötig ein. Ein Beispiel hierfür sind Anforderungen bezüglich einer existierenden Schnittstelle eines anderen Systems oder eines Druckers, die definiert werden, da der Aspekt dem Kontext des Systems zugeordnet und daher nicht veränderbar ist. Würde der Aspekt jedoch (richtigerweise) dem System zugeordnet und somit innerhalb der Systemgrenzen liegen, könnte das System oder der Drucker frei ausgewählt werden bzw. zumindest deren Schnittstelle verändert oder frei gestaltet werden. Anforderungen bezüglich der Schnittstellen würden dann nicht erhoben und dokumentiert.

Liegt ein Aspekt außerhalb der Umfangsgrenze und darf somit nicht verändert werden, aber innerhalb der Systemgrenze (d.h. ist dem System zugeordnet), dann werden für diesen Aspekt keine Anforderungen erhoben, da laut Systemabgrenzung der Aspekt nicht zum Kontext des Systems gehört. Kurz, der Aspekt wird bei der Kontextbetrachtung nicht berücksichtigt und somit werden auch keine Anforderungen bezüglich des Aspekts, wie beispielsweise Schnittstellenanforderungen, erhoben und dokumentiert.

Wir raten Ihnen daher dringend, nur die Systemabgrenzung durchzuführen und mit der Systemgrenze gleichzeitig auch den Umfang des Systems festzulegen und somit wie in diesem Kapitel erläutert, gleichzeitig zu definieren, was verändert bzw. neu gestaltet werden darf und was nicht.

2.1.5 Prinzip 5: Problem · Anforderung · Lösung – Ein unausweichlich ineinandergreifendes Tripel

Das IREB-Glossar definiert ein Problem als:

Definition 2–5: *Problem*

Ein Problem ist eine Schwierigkeit, offene Frage oder ein unerwünschter Zustand, die eine Untersuchung, genauere Beachtung oder eine Lösung benötigen.

Übersetzt aus [IREB-Glossar 2020]

Problem als Ursache für eine Veränderung

In der Regel können sich Probleme auf unterschiedliche Situationen beziehen. Zum Beispiel kann ein Stakeholder unzufrieden sein, weil er mit dem aktuell gegebenen System seine Aufgaben gar nicht oder nicht mit der gewünschten Effektivität bearbeiten kann. Oder im Kontext des Systems (siehe Abschnitt 2.1.4) sind Änderungen eingetreten (Än-

derungen an gesetzlichen Vorgaben, in den unterstützten Geschäftsprozessen etc.), die eine Anpassung des Systems notwendig machen.

Anforderungen: Erster wichtiger Schritt zur Lösung des Problems

Ein Problem kann somit der Ausgangspunkt einer Entwicklung (Neuentwicklung oder Anpassung eines Systems) sein. Der erste Schritt in einer solchen Entwicklung sollte die Betrachtung der Anforderungen sein, die zur Lösung des gegebenen Problems beitragen.

Problemlösung durch soziotechnisches System

Mit der Definition von qualitativ hochwertigen Anforderungen wird aber kein Stakeholder zufrieden sein. Er benötigt eine Realisierung der Anforderungen durch ein System. Die Anforderungen müssen in einem System realisiert werden, mit dessen Hilfe dann die gegebenen Probleme gelöst werden. Ein solches System, das eine Lösung für die Probleme darstellt, ist im Allgemeinen ein soziotechnisches System. Es besteht aus technischen Anteilen (z.B. ein Softwaresystem) und den sozialen Anteilen (die Nutzer der technischen Anteile). In der Regel wird im IT-Umfeld mit dem zu entwickelnden System nur der technische Anteil eines übergreifenden, soziotechnischen Systems verstanden.

Problem-Anforderung-Lösungs-Kette existiert nur in der Theorie.

Zusammenfassend können wir eine Kette, bestehend aus Problem, Anforderung und Lösung, aufbauen. In der Theorie wird ein Problem gegeben, in den Anforderungen wird beschrieben, wie ein System zur Lösung des Problems beitragen soll, und das System löst dann das Problem, indem es die Anforderungen realisiert. Diese Betrachtungsweise muss für die Realität jedoch angepasst werden.

Innovative Lösungen bedingen zusätzliche Anforderungen.

Zum einen ist die Reihenfolge nicht immer so linear vorgegeben. Als Beispiel sei hier die Entwicklung einer Innovation genannt, die aus dem Einsatz einer neuartigen Technologie (also einer neuen Lösung) resultiert. Sie kennen vielleicht solche Sätze wie: »Wenn wir das *SO* machen, dann könnte das System auch noch *DAS* machen.« In anderen Worten: Aus dem Einsatz einer neuen Lösung ergeben sich neue Anforderungen an das System. Und mit diesen Anforderungen können vielleicht zusätzliche Probleme eines bestehenden Systems gelöst werden, die bisher nicht Bestandteil der Überlegungen waren: »Wenn das System *DAS* macht, dann kann der Benutzer *DIESE AUFGABE* besser bearbeiten.«

Anforderungen dokumentieren Problemlösungen.

Zum anderen muss der Begriff »Lösung« nicht auf das fertig implementierte System beschränkt werden. Anforderungen, die aus Problemen oder anderen Anforderungen resultieren, können auch als Lösungen verstanden werden. Als Beispiel betrachten wir die Anforderung an ein Smart-Home-System, Personen identifizieren zu müssen, um z.B. eine Haustür automatisch zu entriegeln. Eine Lösung für diese Anforderung könnte eine Gesichtserkennung mithilfe einer Kamera sein. Diese Lösung kann nun mit Anforderungen gleichgesetzt werden, die, in anderen Worten, die ursprüngliche Anforderung detaillieren (siehe Abschnitt 3.1.3). Dieser Zusammenhang zwischen Anforderungen und Lösungen ist in Abbildung 2–2 noch einmal dargestellt.

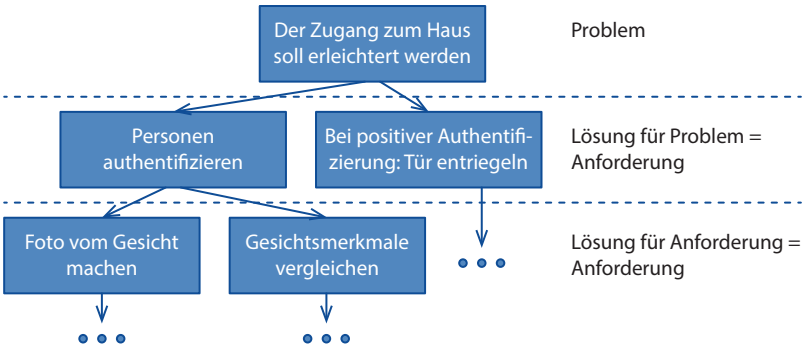


Abb. 2–2
Abhängigkeit zwischen Problem, Anforderung und Lösung

Problem, Anforderung und Lösung sind eng miteinander verflochten.

In jedem Fall sind Problem, Anforderung und Lösung eng miteinander verflochten. Sie können nicht isoliert voneinander betrachtet werden. Beim Denken, Kommunizieren und Dokumentieren sollte der Requirements Engineer dennoch versuchen, diese zu trennen. Die Anforderungen an das System sollten auf keinen Fall die Entwicklung des Systems unnötig einschränken, indem beispielsweise Lösungen, die nicht zwingend nötig sind, gefordert werden. Auch sollte der Requirements Engineer reflektieren, was aus der Vorgabe einer Anforderung für die vorgegebenen Probleme oder für neue Probleme folgt. So könnte in dem obigen Beispiel das Entriegeln der Tür auch in Notsituationen sinnvoll sein.

Kernfakten 2–5: *Problem – Anforderung – Lösung*

Exkurs: Twin-Peaks-Modell

In der Praxis zeigt sich, dass in dem Tripel zwischen Problem, Anforderungen und Lösungen der Architekturschritt eine wichtige Rolle spielt. Oftmals existieren mehrere hierarchische Abstraktionsebenen und auf jeder Ebene gibt es sowohl Anforderungen als auch eine Zerlegung des jeweiligen Betrachtungsgegenstands. Anforderungen auf den tieferen Abstraktionsebenen entstehen aus Architektur- und Designentscheidungen auf der nächsthöheren Ebene und haben somit eine Abhängigkeitsbeziehung zueinander.

Eine schöne Erklärung für diesen Zusammenhang bietet das Twin-Peaks-Modell [Nuseibeh 2001]: Abbildung 2–3 veranschaulicht die Wechselwirkungen zwischen Anforderungen und Architektur für drei Systemebenen.

→

Anforderungen sind der Ausgangspunkt für den Entwurf der Systemarchitektur. Allerdings liefern Architekturentscheidungen neue Erkenntnisse, die sich in Anforderungen niederschlagen müssen. Zudem können Anforderungen durch Architekturentscheidungen ermöglicht oder eingeschränkt werden (siehe [Miller et al. 2008]). Das Twin-Peaks-Modell verdeutlicht, dass in diesem Zusammenspiel eine zunehmende Detailgenauigkeit erreicht wird. Die Anforderungen können auch aus den Architekturentscheidungen folgen und ohne Wechsel in eine tiefere Abstraktionsebene detailliert werden.

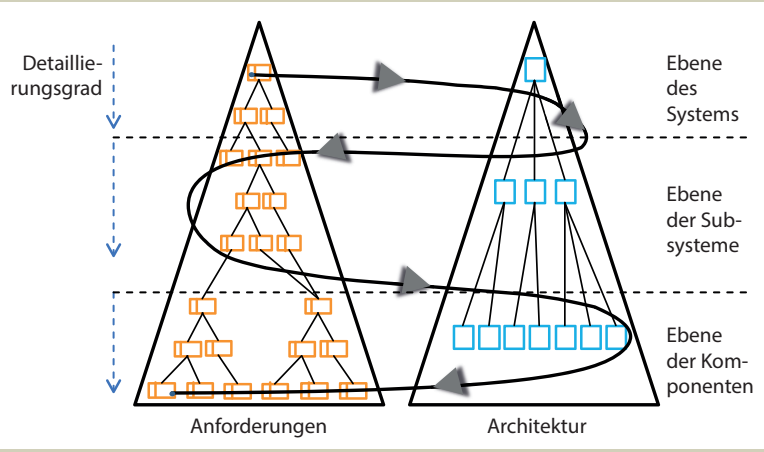


Abb. 2–3
Wechselwirkungen zwischen Requirements Engineering und Architekturentwurf (basierend auf [Nuseibeh 2001] und [Ward und Mellor 1985])

2.1.6 Prinzip 6: Validierung – Nicht validierte Anforderungen sind nutzlos

In Kapitel 1 wurde die Wichtigkeit des Requirements Engineering für die Entwicklung von Systemen erläutert. Gutes Requirements Engineering leistet einen wichtigen Beitrag, dass Entwicklungsprojekte im Zeit- und Budgetrahmen erfolgreich beendet werden können und dass das System die Wünsche und Bedürfnisse der Stakeholder erfüllt (siehe auch Abschnitt 2.1.2).

Qualitätssicherung von Anforderungen ist essenziell!

Der letztgenannte Punkt kann erst am Ende der Realisierung des Systems mit Sicherheit beantwortet werden. Aber auch das Requirements Engineering kann einen wichtigen Beitrag dazu leisten, dass ein System die Wünsche und Bedürfnisse der Stakeholder erfüllt. Der Schlüssel hierzu ist die kontinuierliche Überprüfung während des Requirements Engineering, ob die dokumentierten Anforderungen die Bedürfnisse und Wünsche der Stakeholder widerspiegeln bzw. auf andere Quellen – wie beispielsweise Gesetze – zurückzuführen sind.

Fokus der Validierung von Anforderungen

Hierfür müssen wir die Anforderungen validieren. Bei der Validierung von Anforderungen wird überprüft, inwieweit

a) die Anforderung tatsächlich den Bedürfnissen und Wünschen der Stakeholder entspricht oder auf andere Anforderungsquellen zurückzuführen ist (Abschnitt 4.4),
b) die Anforderung ausreichend detailliert dokumentiert wurde, um dieses Verständnis an die Entwickler zu vermitteln,
c) die Stakeholder sich über die Anforderung einig sind, d.h., dass bezüglich der Anforderung keine ungelösten Konflikte existieren (Abschnitt 4.3), und
d) die über den Kontext des Systems getroffenen Annahmen korrekt und während des Betriebs des Systems gültig sind (Abschnitt 2.1.4).

Kernfakten 2–6: *Validierung*

Exkurs: Validierung

Bei der Validierung von Anforderungen sollten folgende Tipps beachtet werden:

- Bedenken Sie, dass unterschiedliche Stakeholder unterschiedliche Meinungen vertreten. Es reicht also nicht, nur einen Stakeholder zu finden, der einer Anforderung zustimmt. Vielmehr müssen die Anforderungen zwischen den jeweils relevanten Stakeholdern abgestimmt sein.
- Berücksichtigen Sie bei der Validierung der Anforderung, dass Sie diese u.a. hinsichtlich der Bedürfnisse und Wünsche der Stakeholder prüfen. Stakeholder sind Menschen!
- Berücksichtigen Sie, dass sich die Bedürfnisse und Wünsche der Stakeholder ändern, worauf wir in Prinzip 7 noch näher eingehen werden. Dies kann unterschiedliche Ursachen haben. Dies mag durch technologischen Fortschritt, durch Wissensgewinn oder auch durch die Kommunikation mit anderen Stakeholdern oder dem Requirements Engineer selbst ausgelöst werden. Daher ist es wichtig, Anforderungen regelmäßig zu validieren.

→

- Denken Sie daran, dass die Wünsche und Bedürfnisse nur durch zielgerichtete Kommunikation mit den Stakeholdern erhoben werden können. Hierfür sind die Verfügbarkeit der Stakeholder und die dadurch entstehenden Kosten zu planen. Weiterhin muss der Requirements Engineer Gefahren kennen, die durch »Information Hiding« entstehen können, und geeignete Maßnahmen ergreifen, um dennoch die wichtigen Informationen zu erlangen. Bedenken Sie jedoch, dass es in der Praxis oft ganz banale Gründe sind, warum Ihnen Stakeholder nicht die benötigten Informationen zur Verfügung stellen. Oft wissen die Stakeholder nicht, dass bestimmte Informationen für den Requirements Engineer wichtig sind.
- Auch andere Anforderungsquellen, wie zum Beispiel Gesetzestexte, werden sich ändern (siehe Abschnitt 2.1.7). Um eine Konformität des Systems sicherzustellen, sollten Sie die Anforderungen auch diesbezüglich validieren.
- Beginnen Sie früh mit der Validierung von Anforderungen. Kontinuierliche Validierung hilft, Anforderungen iterativ immer weiter zu verbessern und die Qualität des Endprodukts sicherzustellen.

2.1.7 Prinzip 7: Evolution – Sich ändernde Anforderungen sind kein Unfall, sondern der Normalfall

Änderungen von Anforderungen sind unvermeidbar.

Die Welt entwickelt sich weiter, und Anforderungen tun das auch. Eine Weiterentwicklung von Anforderungen bedeutet im Allgemeinen, dass die Anforderungen geändert oder erweitert werden. Beides definieren wir als »Änderung von Anforderungen«.

Gründe für Anforderungsänderungen

Unter anderem lösen die folgenden Ereignisse Änderungen an Anforderungen aus:

- Sich ändernde Wünsche, Bedürfnisse, Ideen und Prioritäten von Stakeholdern
- Änderung von Geschäftsprozessen, die das System betreffen
- Verfügbarkeit von neuen Produkten und Lösungen von Wettbewerbern am Markt
- Technologische Veränderungen und Neuerungen
- Änderungen von Gesetzen, Vorschriften und Rahmenbedingungen
- Feedback von Stakeholdern (z. B. bei der Validierung oder Abnahme)

- Aufdecken von Fehlern oder Widersprüchen in bestehenden Anforderungen
- Feedback von aktuellen Systemnutzern beispielsweise Fehlermeldungen oder neue Bedürfnisse

Alle diese Ereignisse treten in der Praxis auf. Änderungen von Anforderungen sind daher in der Realität der Normalfall! Sie können Änderungen zwar minimieren, aber keinesfalls generell vermeiden.

Zwickmühle: Veränderung und Stabilität

Ein gutes Requirements Engineering sollte folgende, scheinbar widersprüchliche Ziele erreichen:

- Zulassen, dass sich Anforderungen ändern können, da eine Nichtanpassung der Anforderungen an die veränderte Situation in der Realität zu Problemen führen würde.
- Anforderungen (möglichst) stabil halten, da Änderungen an Anforderungen oftmals mit zum Teil erheblichen Kosten und eventuellen Verschiebungen von Fristen verbunden sind. Zudem ist eine Systementwicklung mit sich ständig stark verändernden Anforderungen kaum möglich.

Ein sinnvolles, den Randbedingungen angepasstes Vorgehen im Requirements Engineering (siehe Kap. 5) unterstützt dabei, einerseits mit den Änderungen adäquat umzugehen und andererseits Defizite in den Anforderungen, die zu Änderungen führen würden, früh zu erkennen und zu beheben.

Kernfakten 2–7: *Evolution*

Exkurs: Änderungen vs. Stabilität

Anforderungsänderungen sind nicht vermeidbar, können aber durch entsprechende Maßnahmen minimiert und beherrschbar werden. Die folgenden beiden Vorgehensweisen unterstützen Sie, um einerseits Veränderungen der Anforderungen zuzulassen, ja sogar zu ermutigen, andererseits aber die Anforderungen für den weiteren Entwicklungsprozess stabil zu halten:

→

- Kurze Entwicklungszyklen, innerhalb derer die Anforderungen stabil bleiben: Am Ende eines Zyklus können dann die gewünschten Änderungen bewertet und neu eingeplant werden. So landen alle Änderungswünsche bei agilen Vorgehensmodellen nicht im aktuellen Sprint, sondern im Product Backlog, werden dort priorisiert und in einem der folgenden Sprints eingeplant. Mehr zu der Konfiguration des Requirements Engineering für kurze Entwicklungszyklen finden Sie in Abschnitt 5.3.
- Change Management für Anforderungen zusammen mit einem guten Konfigurationsmanagement: Für ein gutes Änderungsmanagement spielen folgende Tätigkeiten aus dem Anforderungsmanagement eine wichtige Rolle (siehe auch Abschnitt 6.7).
  - Die Attribuierung der Anforderungen ermöglicht es z.B., eine Anforderung einem bestimmten Systemrelease zuzuordnen.
  - Die Versionskontrolle ermöglicht eine Unterscheidung von verschiedenen Versionen von Anforderungen.
  - Die Verfolgbarkeit von Anforderungen ermöglicht es, Auswirkungen von Änderungen zu identifizieren und entsprechend zu berücksichtigen.

In vielen Projekten werden Änderungen in Change Requests (Änderungsanträgen) dokumentiert und durch ein Change Control Board geprüft und freigegeben oder abgelehnt.

2.1.8 Prinzip 8: Innovation – Mehr vom Gleichen ist nicht genug

Limitation von Stakeholder-Befragungen

Befragt man Stakeholder im Rahmen eines Interviews oder mit Fragebögen, äußern sie meist nur die Wünsche und Bedürfnisse, an die sie bewusst denken. Die unbewussten (das, was ihnen noch nicht bekannt ist und z.B. bei der Benutzung des Systems überrascht) oder unterbewussten Wünsche und Bedürfnisse (das, was der Stakeholder als bekannt voraussetzt oder ihm gar nicht bewusst ist) gehen dabei oft unter. Ein guter Requirements Engineer hat Praktiken zur Verfügung, alle Arten von Wünschen und Bedürfnissen der Stakeholder zu ermitteln. Er strebt danach, nicht nur das zu erfüllen, was der Stakeholder äußert, sondern auch das, was er nicht äußert.

Innovation im Kleinen und Großen

Bestimmte Requirements-Engineering-Praktiken fördern Innovation im Kleinen und im Großen: im Kleinen durch das Streben nach neuen aufregenden Funktionen und nach der Verbesserung des Systems, z.B. bezüglich der Benutzerfreundlichkeit oder der Leistung, im Großen durch das Streben nach disruptiven neuen Ideen.

Inkrementelle vs. disruptive Innovation

Grundsätzlich lässt sich daher zwischen inkrementellen und disruptiven Innovationen unterscheiden. Von einer inkrementellen Innovation spricht man dann, wenn ein bestehendes Produkt oder eine Technologie stetig verbessert wird, sodass diese(s) z. B. effizienter oder günstiger wird oder eine neue Funktionalität hinzugefügt wird. Von einer disruptiven Innovation spricht man, wenn der Markt sich im Nutzungsverhalten so stark verändert, dass bekannte Technologien oder Produkte durch neue Technologien oder Produkte vollständig abgelöst und sozusagen damit »zerstört« werden. Zum Beispiel wurde durch die Einführung des MP3-Players der Discman (portabler CD-Spieler) fast vollständig abgelöst.

Innovationen durch Kreativitätstechniken fördern

Innovative Ideen können nicht durch Befragung der Stakeholder ermittelt werden. Hierfür müssen andere Ermittlungstechniken, die Kreativitätstechniken, eingesetzt werden. Beispiele für solche Kreativitätstechniken sind Design Thinking oder Designing for Growth, die in Workshops mit den Stakeholdern eingesetzt werden können. Kreativitätstechniken fördern das Maß an Kreativität und erleichtern es, das Denken in herkömmlichen Bahnen aufzubrechen. Welche Ermittlungstechniken im Rahmen des Requirements Engineering zu welchem Zweck angewendet werden sollten, wird in Abschnitt 4.2 erläutert.

Kernfakten 2–8: *Innovation*

2.1.9 Prinzip 9: Systematische und disziplinierte Arbeit – im Requirements Engineering unverzichtbar

Kein »one fits all« im Requirements Engineering

Nicht jede Systementwicklung läuft nach dem gleichen Schema ab. Sehr oft unterscheiden sich die Anwendungsdomäne, die Problemstellung, der Einsatz des Systems, die beteiligten Personen, die Arbeitsumgebung und viele weitere Charakteristika eines Entwicklungsprojekts. Für den Requirements-Engineering-Prozess gibt es daher kein »one fits all«. Es müssen verschiedene Praktiken und Techniken für die Ausführung der typischen Tätigkeiten des Requirements Engineering (Ermitteln, Dokumentieren, Übereinstimmung erzielen, Validieren und Verwalten von Anforderungen) situativ ausgewählt, in Zusammenhang gestellt und letztendlich auch eingesetzt werden.

Situative Auswahl der auszuführenden Aktivitäten

Unabhängig von dem für die Systementwicklung ausgewählten Vorgehensmodell (bspw. wasserfallähnlich oder agil) sollte das Requirements Engineering systematisch durchgeführt werden. Abhängig von

der Problemstellung, dem aktuellen Prozesszustand, dem Kontext, den beteiligten Stakeholdern, aber auch von dem angestrebten Arbeitsprodukt sollten die richtigen Prozesse, Praktiken und Arbeitsprodukte ausgewählt werden, die entsprechend der durch das Projekt gegebenen Situation am erfolgreichsten eingesetzt werden können. Dabei sollte auf Erfahrungen aus früheren, vergleichbaren Projekten zurückgegriffen, aber auch die zuvor eingesetzten Prozesse und Praktiken nicht unreflektiert übernommen werden.

Systematisches Vorgehen auch in agilen Prozessen

Eine agile Vorgehensweise gilt hierbei nicht als Ausrede, d.h., Agilität ist kein Grund, das Requirements Engineering unsystematisch und ad hoc durchzuführen. Gerade agile Projekte dürfen nicht dazu verleiten, durch die in einer Iteration auftauchenden neuen Anforderungen oder durch Änderungen in den Anforderungen bereits bekannte Anforderungen zu ignorieren oder diese nicht ausreichend zu berücksichtigen.

Mehr zu der Konfiguration des Requirements-Engineering-Prozesses finden Sie in Kapitel 5.

Kernfakten 2–9: *Systematische und disziplinierte Arbeit*

Exkurs: Auswahl von Praktiken und Techniken

Systematisches Vorgehen heißt, eine Arbeit planmäßig und konse-quent durchzuführen. Und das gilt auch für die Auswahl der geeig-neten Praktiken und Techniken.

Jedes Projekt benötigt einen systematisch entwickelten, passgenauen Umgang mit Anforderungen. Um dies zu erreichen, ist Folgendes erforderlich:

- Der Requirements Engineer analysiert das Projekt und seine Randbedingungen und berücksichtigt die Auswirkungen der Requirements-Engineering-Tätigkeiten auf alle Teile des Projekts (auch soziale Komponenten).
- Er wendet verschiedene Praktiken abhängig von der Anwendungsdomäne, der Problemstellung, der aktuellen Prozesssituation und den beteiligten Personen an.
- Er reflektiert erfolgreich verwendete Praktiken und Prozesse aus vorangegangenen Projekten und verwendet diese nicht unreflektiert.

→

- Er entwickelt einen »Praktiken-Baukasten« von Projekt zu Projekt weiter, um den positiven und negativen Aspekten der bisherigen Arbeiten Rechnung zu tragen.

Nach der Anwendung einer Praktik bzw. Technik ist es ratsam, über die Verwendung und Auswahl der Technik zu reflektieren. Dies kann der Requirements Engineer für sich tun – besser ist es jedoch, die Personen einzubeziehen, die jeweils bei der Verwendung der Praktik bzw. Technik involviert waren.

In agilen Vorgehensweisen (z.B. Scrum, Kanban) erfolgt eine solche Reflektion, indem das Team gemeinsam über den letzten Sprint reflektiert und versucht, Maßnahmen zur Verbesserung der Teamzusammenarbeit und der eingesetzten Techniken zu finden. Bei klassischen Vorgehensweisen können durchaus explizite Reflektionstreffen eingeführt werden, beispielsweise nach Abschluss eines Meilensteins und/oder nach Bedarf.

2.2 System und Kontextabgrenzung

In diesem Abschnitt gehen wir auf die Tätigkeiten eines Requirements Engineer ein, die zur Erfüllung des Prinzips 4 (Abschnitt 2.1.4), durchgeführt werden sollten. Ein Requirements Engineer sollte das System von dessen Umgebung abgrenzen und den Teil der Umgebung identifizieren, der die Anforderungen an das zu entwickelnde System bestimmt. Ein über diesen Abschnitt hinausgehendes tieferes Verständnis der Abgrenzung von System und Kontext findet sich in [Pohl 2010].

2.2.1 Systemkontext

Antizipieren des Systems im Betrieb

Dem Requirements Engineering kommt die Aufgabe zu, in der Umgebung des geplanten Systems alle diejenigen materiellen und immateriellen Aspekte zu identifizieren, die eine Beziehung zu dem System haben. Hierzu wird eine Sollperspektive eingenommen, d.h., es wird eine Annahme getroffen, wie das geplante System später in die Umgebung integriert werden soll. Hierdurch wird der Realitätsausschnitt identifiziert, der das System und damit potenziell auch dessen Anforderungen beeinflusst. Um die Anforderungen an das geplante System korrekt und vollständig spezifizieren zu können, ist es notwendig, die Beziehungen zwischen den einzelnen materiellen und immateriellen Aspekten im Systemkontext und dem geplanten System exakt zu identifizieren. Der für die Anforderungen des Systems relevante Ausschnitt der Realität wird als Systemkontext bezeichnet.

Kontextaspekte im Systemkontext

Zu den möglichen Typen von Aspekten im Systemkontext gehören u.a.:

- Personen (Stakeholder oder Stakeholder-Gruppen)
- Systeme im Betrieb (andere technische Systeme oder Hardware)
- Alt-/Vorgängersysteme
- Prozesse (technisch oder physikalisch, Geschäftsprozesse)
- Ereignisse (technisch oder physikalisch)
- Dokumente (z.B. Gesetze, Standards, Systemdokumentationen)

Abbildung 2–4 zeigt skizzenhaft den Systemkontext eines Systems, der aus Systemen und Stakeholdern, die mit dem zu entwickelnden System an dessen Schnittstelle eine Nutzungsbeziehung eingehen, und aus weiteren Quellen von Anforderungen besteht.

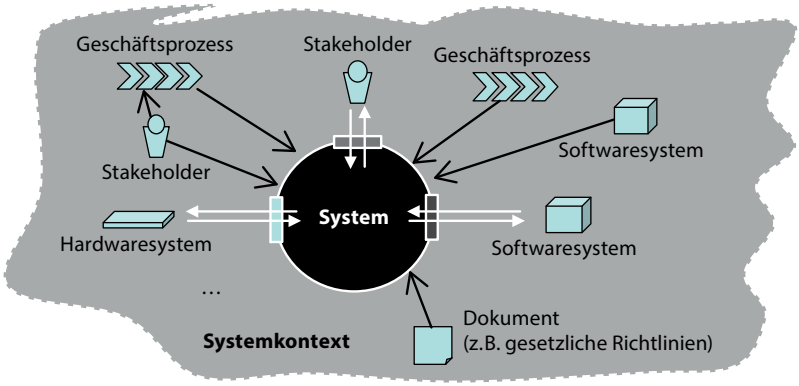


Abb. 2–4
Arten von Aspekten im Systemkontext

Konsequenzen fehlerhafter oder unvollständiger Kontextberücksichtigung

Wird der Systemkontext im Rahmen des Requirements Engineering inkorrekt oder unvollständig berücksichtigt, hat dies unvollständige oder fehlerhafte Anforderungen zur Konsequenz. Dies führt dazu, dass das entwickelte System auf der Grundlage unvollständiger oder fehlerhafter Annahmen arbeitet, was eine häufige Ursache für Systemversagen im Betrieb ist. Solche Fehler bleiben bei der Überprüfung, ob das zu entwickelnde System den spezifizierten Anforderungen genügt, häufig unentdeckt und treten erst später im Betrieb des Systems auf, und das mit teilweise katastrophalen Folgen.

Systemkontext und Anforderungskontext

Der Ursprung der Anforderungen eines Systems liegt im Systemkontext des geplanten Systems. Beispielsweise fordern Stakeholder, einschlägige Standards oder gesetzliche Richtlinien bestimmte funktionale Merkmale oder Qualitäten, die das zu entwickelnde System an dessen Schnittstelle aufweisen soll. Eine Anforderung ist also für einen spezifischen Kontext definiert und kann auch nur für diesen Kontext richtig interpretiert werden. Je vollständiger der Kontext eines Systems und damit der Ursprung einer Anforderung bekannt ist, umso geringer ist

die Wahrscheinlichkeit für eine falsche Interpretation dieser Anforderung. Daher ist eine zweckmäßige Dokumentation des Systemkontexts bzw. der Informationen über den Systemkontext für den Entwicklungsprozess von besonderer Bedeutung.

2.2.2 System- und Kontextgrenzen bestimmen

Die Bestimmung des Systemkontexts liegt in der Verantwortung des Requirements Engineering. Um den Systemkontext zu definieren, ist es notwendig, diesen sowohl von dem zu entwickelnden System als auch vom irrelevanten Teil der Realität abzugrenzen.

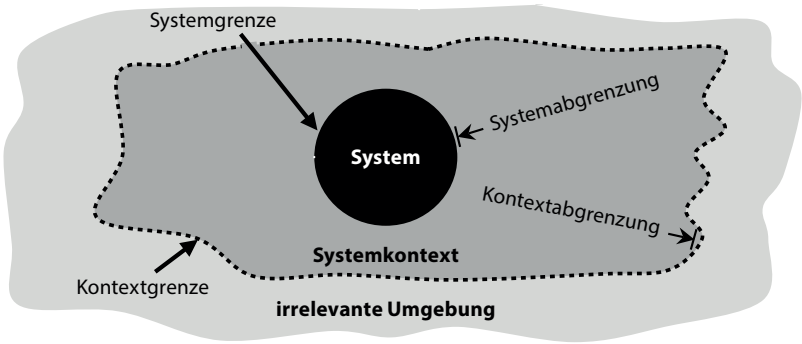


Abb. 2–5
Systemgrenze und Kontextgrenze eines Systems

Zur Abgrenzung des Systemkontexts wird daher zwischen zwei Abgrenzungsprozessen unterschieden (vgl. Abb. 2–5):

- *Systemabgrenzung*
  Im Rahmen der Systemabgrenzung wird die Systemgrenze bestimmt, die festlegt, welche Aspekte durch das geplante System abgedeckt werden sollen und welche Aspekte Teil der Umgebung dieses Systems sind.
- *Kontextabgrenzung*
  Im Rahmen der Kontextabgrenzung wird die Grenze des Kontexts zur irrelevanten Umgebung hin bestimmt, indem analysiert wird, welche Aspekte in der Umgebung eine Beziehung zu dem geplanten System haben.

Systemgrenze und Kontextgrenze definieren den Systemkontext.

Durch Systemgrenze und Kontextgrenze wird der Systemkontext bestimmt. Der Systemkontext umfasst alle Aspekte, die für die Anforderungen des geplanten Systems relevant sind und nicht im Rahmen der Entwicklung dieses Systems gestaltet werden können (solche Aspekte liegen innerhalb der Systemgrenze).

Die Systemgrenze festlegen

Die Systemgrenze grenzt den Konstruktionsgegenstand zur Umgebung hin ab. Durch die Wahl der Systemgrenze wird festgelegt, welche Aspekte das zu konstruierende System abdeckt und was außerhalb des Systems liegt. Sämtliche Aspekte, die innerhalb der Systemgrenzen liegen, können somit im Systementwicklungsprozess verändert bzw. gestaltet werden. Innerhalb der Systemgrenzen kann sich z. B. ein aus Hardware und Software bestehendes System befinden, das durch das geplante System ersetzt werden soll. Aspekte im Systemkontext können auch Geschäftsprozesse, technische Prozesse, Personen, Organisationsstrukturen und Bestandteile der IT-Infrastruktur sein.

Quellen und Senken als Ausgangspunkt der Abgrenzung

Zur Identifikation der Schnittstellen des Systems mit der Umgebung können u. a. die Quellen und Senken des geplanten Systems betrachtet werden (z. B. [DeMarco 1978]). Quellen liefern Eingaben für das System. Senken erhalten Ausgaben vom System. Mögliche Quellen und Senken eines Systems sind:

- Stakeholder(-Gruppen)
- Existierende Systeme (technische oder nicht technische Systeme)

Schnittstellen: Interaktion zwischen System und Umgebung

Quellen und Senken interagieren mit dem späteren System über die Systemschnittstellen. Mittels der Systemschnittstellen stellt das System im Betrieb der Umgebung seine Funktionalität zur Verfügung, überwacht die Umgebung, beeinflusst Umgebungsparameter oder steuert Abläufe in der Umgebung. Abhängig vom Typ der jeweiligen Quelle oder Senke benötigt das geplante System verschiedenartige Schnittstellen (z. B. Mensch-Maschine-Schnittstelle, Hardwareschnittstelle oder Softwareschnittstelle), wobei der Schnittstellentyp wiederum spezifische Randbedingungen oder zusätzliche Quellen von Anforderungen an das geplante System implizieren kann.

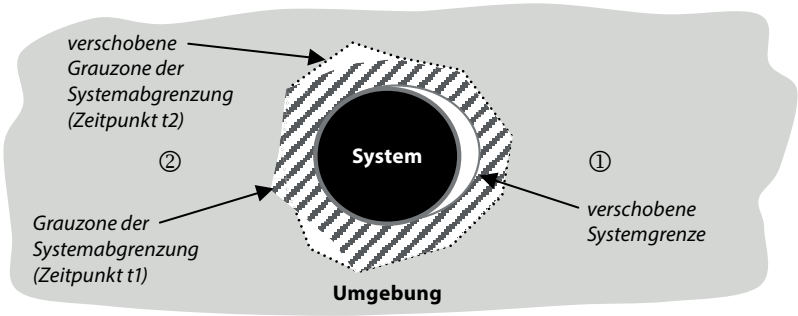
Grauzone der Systemabgrenzung

Die Systemgrenze ist häufig erst gegen Ende des Requirements-Engineering-Prozesses präzise festgelegt. Zuvor sind einige oder mehrere Schnittstellen bzw. gewünschte Funktionen und Qualitäten des geplanten Systems nur unvollständig oder überhaupt nicht bekannt. Diese zeitweise Unschärfe der Systemabgrenzung führt dazu, dass zu bestimmten Zeitpunkten im Requirements-Engineering-Prozess eine Grauzone in der Systemabgrenzung besteht (vgl. Abb. 2–6). Zum Beispiel könnte zu einem frühen Zeitpunkt im Requirements-Engineering-Prozess noch nicht geklärt sein, ob das System eine bestimmte Funktion (z. B. »Zahlungsabwicklung per Kreditkarte«) implementieren soll oder ob ein anderes System im Systemkontext eine entsprechende Funktion anbietet, die genutzt werden soll (z. B. »System zur Zahlungsabwicklung«).

Verschiebung der Grauzone

Neben einer Verschiebung der Systemgrenze innerhalb der Grauzone (① in Abb. 2–6) kann sich im Laufe des Requirements Engineering auch die Grauzone der Systemabgrenzung selbst verschieben (② in Abb. 2–6). Eine solche Verschiebung wird beispielsweise dadurch verursacht, dass Aspekte, die vorher zum Systemkontext gehörten, im Rahmen der Entwicklung des geplanten Systems möglicherweise nun doch verändert werden sollen und damit in die Systemgrenze wandern. Dies trifft z.B. dann zu, wenn in Bezug auf einen Geschäftsprozess im Systemkontext die Situation eintritt, dass für bestimmte Aktivitäten des Geschäftsprozesses nicht mehr feststeht, ob diese von dem zu entwickelnden System verändert werden dürfen oder nicht. Das heißt, man weiß nicht, ob diese Aktivitäten dem System zugerechnet werden sollten (und somit veränderbar sind) oder doch dem Systemkontext zugeordnet werden sollten (und somit nicht veränderbar wären und vom System unterstützt werden sollen). Hierdurch ändert sich die Grauzone in der Systemabgrenzung entsprechend (vgl. Abb. 2–6).

Abb. 2–6
Grauzone der Systemabgrenzung



Die Kontextgrenze bestimmen

Relevante Kontextobjekte

Die Kontextgrenze differenziert in der Umgebung des geplanten Systems zwischen Kontextaspekten, d.h. Aspekten der Umgebung, die im Requirements Engineering (z.B. als Anforderungsquellen) berücksichtigt werden müssen, und Aspekten, die für das geplante System irrelevant sind.

Konkretisierung und Verschiebung der Kontextgrenze

Zu Beginn eines Requirements-Engineering-Prozesses sind häufig nur ein Teil der Umgebung sowie einzelne spezifische Beziehungen zwischen der Umgebung und dem geplanten System bekannt. Über den Verlauf des Requirements Engineering hinweg ist es erforderlich, die Grenze zwischen Systemkontext und irrelevanter Umgebung zu konkretisieren, indem relevante Aspekte der Umgebung im Hinblick auf Beziehungen zum geplanten System analysiert werden. Neben der Systemgrenze verschiebt sich für gewöhnlich im Laufe des Requirements Engineering auch die Kontextgrenze. Wird z.B. festgestellt, dass eine

vormals als relevant eingestufte gesetzliche Vorschrift wider Erwarten keinerlei Auswirkungen auf das geplante System hat bzw. aus einem bestimmten Grund die Relevanz verliert, reduziert sich der Systemkontext entsprechend (① in Abb. 2–7). Wird eine neue Vorschrift identifiziert, die Auswirkungen auf das zu entwickelnde System hat, erweitert sich der Systemkontext (② in Abb. 2–7).

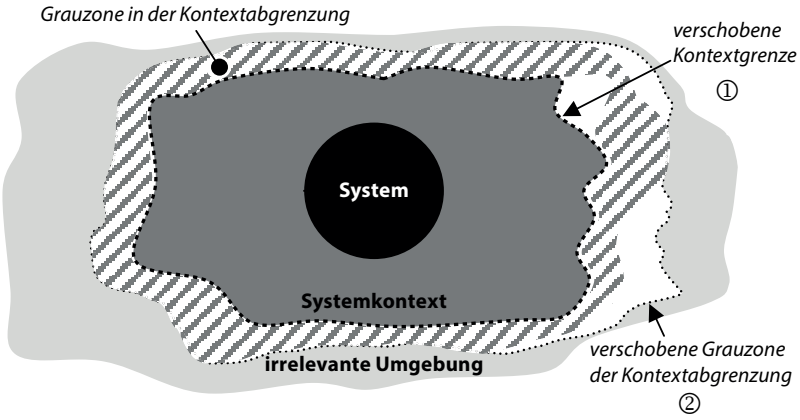


Abb. 2–7
Grauzone der Kontextabgrenzung

Da die Kontextgrenze den Systemkontext von der gesamten irrelevanten Umgebung abgrenzt, ist eine vollständige und präzise Kontextabgrenzung für komplexe Systeme praktisch nicht möglich. Zudem kann eventuell für einzelne Aspekte in der Umgebung des geplanten Systems nicht festgestellt werden, ob diese Aspekte das geplante System beeinflussen bzw. von diesem beeinflusst werden. Diese beiden Beobachtungen sind ursächlich dafür, dass, ähnlich wie bei der Systemabgrenzung, auch in Bezug auf die Kontextgrenze eine Grauzone existiert (vgl. Abb. 2–7).

Auflösung der Grauzonen

Die Grauzone der Kontextabgrenzung umfasst somit identifizierte Aspekte der Umgebung, für die unklar ist, ob sie eine Beziehung zum geplanten System haben oder nicht. Im Gegensatz zur Grauzone in der Systemabgrenzung, die im Laufe des Requirements Engineering aufgelöst werden muss, ist es nicht erforderlich, die Grauzone in der Kontextabgrenzung vollständig aufzulösen.

Exkurs: Dokumentation des Systemkontexts

Um den Systemkontext eines Systems möglichst vollständig zu dokumentieren, werden in der Regel verschiedene Dokumentationsformen eingesetzt. Zur Dokumentation des Systemkontexts werden oftmals Use-Case-Diagramme [Jacobson et al. 1992] oder Datenflussdiagramme [DeMarco 1978] eingesetzt. Bei der Kontextmodellierung auf der Basis von Datenflussdiagrammen werden die Quellen und Senken in der Umgebung des Systems modelliert, die Ursprung oder Endpunkt von Datenflüssen (bzw. Materialflüssen, Energieflüssen, Geldflüssen etc.) zwischen dem betrachteten System und der Umgebung darstellen. In Use-Case-Diagrammen werden die Akteure (z.B. Personen oder andere Systeme) in der Umgebung des Systems und deren Nutzungsbeziehungen mit dem zu entwickelnden System modelliert.