3 Vorgehen bei der Architekturentwicklung

Erfahrung ist die härteste Lehrerin.
Sie gibt Dir zuerst den Test und anschließend den Unterricht.

Susan Ruth, 1993

Fragen, die dieses Kapitel beantwortet:

- Wie sollten Softwarearchitekten vorgehen?
- Was sind die typischen Aufgaben von Softwarearchitekten?
- Wie entwickeln Sie eine „erste Vorstellung" vom System?
- Wie finden Sie die relevanten Einflussfaktoren?
- Wie berücksichtigen Sie diese Einflüsse beim Entwurf von Architekturen?
- Wie erreichen Sie Qualität (nichtfunktionale Anforderungen)?
- Welche typischen Risiken drohen bei Softwarearchitekturen?
- Welche Lösungsstrategien adressieren diese Risiken?

Wie sollen Architekten vorgehen?

Die schlechte Nachricht zuerst: Es gibt kein deterministisches Verfahren, das in jedem Fall zu guten Softwarearchitekturen führt.[1] Sie werden sich praktisch immer iterativ-inkrementell an die Lösung heranarbeiten müssen.

Und jetzt die gute Nachricht: Bild 3.1 zeigt einige grundlegende Aktivitäten, die Ihnen beim effektiven Entwurf von Architekturen helfen.

Diese Aktivitäten beziehen sich sowohl auf die Neuentwicklung als auch auf die Änderung (= Wartung, Evolution, Sanierung) von Systemen. Auf einige Besonderheiten von „Änderungs- oder Wartungsfällen" gehe ich in Abschnitt 3.6 bzw. Kapitel 9 ein.

Konzepte und Ideen wiederverwenden

Sie sollten sich in jedem Fall über mögliche Wiederverwendung informieren: Wer hat eine ähnliche Aufgabe vor Ihnen gelöst und wie? Sammeln Sie Lösungsideen, anstatt immer neu zu entwerfen.[2]

[1] Architekturen und Systeme würden dann automatisch generiert und Sie hätten dieses Buch nicht gekauft.

[2] Bei der Wiederverwendung helfen Ihnen Architekturmuster und –stile (finden Sie in Kapitel 4) sowie technische Konzepte (Kapitel 7).

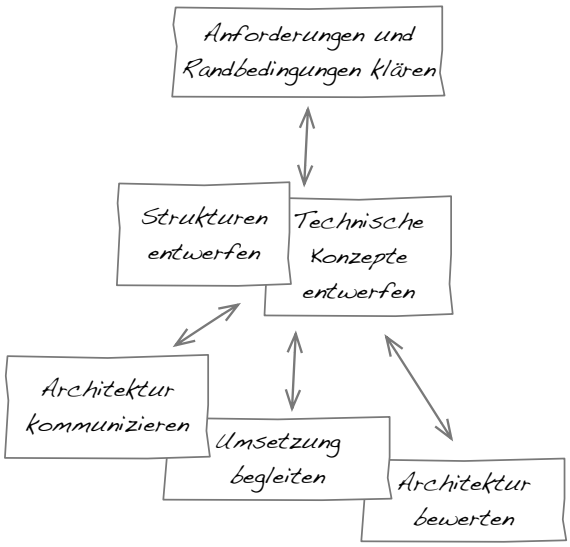

Bild 3.1
Systematischer Entwurf von Softwarearchitekturen

- Klären Sie Anforderungen mit den *maßgeblichen* Beteiligten. Im Idealfall liegen Ihnen dazu aktuelle, inhaltlich korrekte und präzise Beschreibungen aus dem *Requirements Engineering* vor. Siehe Abschnitt 3.2.
- Identifizieren Sie projektspezifische Randbedingungen und Einflussfaktoren (= Einschränkungen). Diese bilden *Leitplanken* Ihrer Entwurfsentscheidungen. Aus Anforderungen und Einflussfaktoren leiten Sie Risiken ab, die den Erfolg Ihrer Architekturarbeit bedrohen. Siehe Abschnitt 3.3.
- Nun treffen Sie (am besten in einem kleinen Team) Entwurfsentscheidungen über Strukturen und Konzepte. Entwickeln Sie Lösungsideen und -ansätze auf Basis der Anforderungen und Randbedingungen. Validieren Sie Entwürfe und Entscheidungen möglichst frühzeitig durch konkrete Implementierung. Hierzu finden Sie weitere Informationen in den Kapiteln 4 und 7.
  - Entwerfen Sie Strukturen: Welche Bausteine gibt es, wie arbeiten sie zusammen, wie und wo laufen sie ab? Diese Strukturentscheidungen folgen grundlegenden Entwurfsprinzipien, „*Best Practices*“ und Heuristiken. Außerdem sollten Sie Architekturstile und -muster, Referenzarchitekturen oder ähnliche Vorlagen zurate ziehen. Häufig beginnen Sie mit fachlichen Strukturen und erweitern diese dann sukzessive um technische Details. Genaueres lernen Sie in Kapitel 4 (Strukturentwurf) kennen.
  - Entwerfen Sie übergreifende (technische) Konzepte wie Persistenz, Benutzeroberfläche, Verteilung, Sicherheit u. a. Das gesamte Kapitel 7 dieses Buchs unterstützt Sie dabei. Das geschieht in der Regel parallel zum Entwurf der Strukturen – daher stehen diese beiden in Bild 3.1 zusammen.
- Kommunizieren Sie Ihre Softwarearchitekturen bedarfsgerecht und angemessen. In Kapitel 5 stelle ich Ihnen dazu einige Sichten vor, die spezifische Interessen der verschiedenen Projektbeteiligten berücksichtigen. Kommunizieren Sie sowohl schriftlich als auch mündlich!

- Während der Implementierung sollten Sie kontinuierlich Rückmeldungen Ihres Entwicklungsteams einholen. Überwachen Sie, ob Ihre Entscheidungen angemessen umgesetzt werden beziehungsweise die von Ihnen gewünschten Auswirkungen zeigen. Ein paar Tipps dazu finden Sie in Abschnitt 3.5.
- Schließlich sollten Sie sich regelmäßig fragen, ob Ihre Architektur den Anforderungen Ihrer Kunden genügt – sowohl in funktionaler wie in qualitativer Hinsicht. Kapitel 8 zeigt Ihnen den Weg zur systematischen Bewertung Ihrer Architekturen. Auch daraus leiten Sie Risiken ab – die Ihnen wiederum Anregungen für konkrete Entwurfsentscheidungen geben.

Architekturen entstehen iterativ und inkrementell

Architektur entsteht iterativ

Einige Aktivitäten aus Bild 3.1 können parallel stattfinden – in Ihrem Kopf oder auf verschiedene Personen verteilt. In jedem Fall klären Sie regelmäßig, welche Anforderungen und Einflussfaktoren sich geändert haben oder welche neuen Erkenntnisse das Entwicklungsteam gewonnen hat – daraufhin müssen Sie eventuell Ihre Entscheidungen überarbeiten und die Architektur Ihres Systems anpassen.

Gehen Sie inkrementell vor, d. h., treffen Sie neue, zusätzliche Entscheidungen auf der Grundlage früherer Erfahrungen und vorhandener Strukturen, Konzepte und Implementierung. Auf diese Weise kann Ihr System auf der Basis von Feedback und konkreter Erfahrung *wachsen*.

Ich möchte Ihnen das iterative Vorgehen bei der Architekturentwicklung noch aus einer weiteren Perspektive verdeutlichen (als ergänzende Erklärung zu Bild 3.1):

Architekturen müssen sowohl fachliche Anforderungen wie auch Qualitätsanforderungen erfüllen. Sie können sich als Architekt oder Entwicklungsteam diesem Problem aus unterschiedlichen Richtungen annähern:

- top-down: vom Abstrakten zum Detaillierten,
- bottom-up: von Details oder feingranularen Teilen zum Abstrakten,
- outside-in: ausgehend von externen Schnittstellen das Innere des Systems gestalten.

Bild 3.2 zeigt eine solche Perspektive auf iteratives Vorgehen bei der Architekturentwicklung:

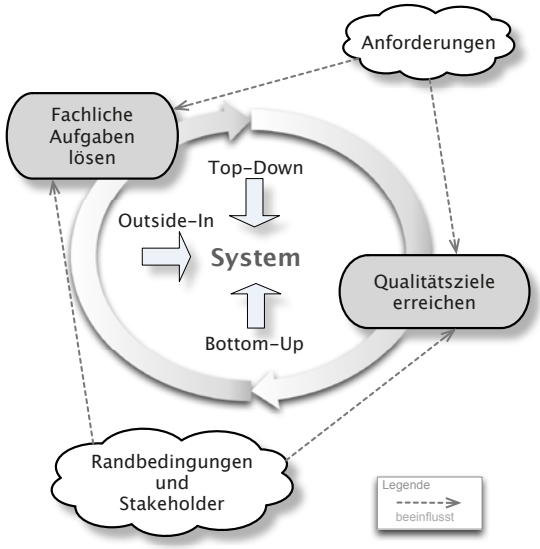

Bild 3.2
Fachliche und Qualitätsziele iterativ erreichen

Architekturen entstehen (meistens) im Team

Softwarearchitekten sollten meiner Ansicht nach mit (überschaubaren) Teams entscheiden, um einerseits das Wissen dieser Teams zu nutzen, andererseits die Akzeptanz von Entscheidungen frühzeitig zu sichern.

Verabschieden Sie sich von dem Gedanken, Softwarearchitekturen „ein für alle Mal“ zu entwickeln. Arbeiten Sie iterativ. Entwickeln Sie Ihre Entwürfe in Zyklen. Bleiben Sie agil und nehmen Sie Rückkopplungen der Projektbeteiligten und der Organisation in Ihre Entwürfe auf. So entstehen Architekturen, die an den wirklichen Bedürfnissen und Anforderungen orientiert sind!

Bild 3.3 zeigt schematisch, dass sowohl Ihre Kunden wie auch andere Projektbeteiligte Anforderungen und Einflussfaktoren ändern. Aufgrund dieser Änderungen müssen Sie Ihren Architekturentwurf und Ihre Architekturentscheidungen iterativ den veränderten Gegebenheiten anpassen.

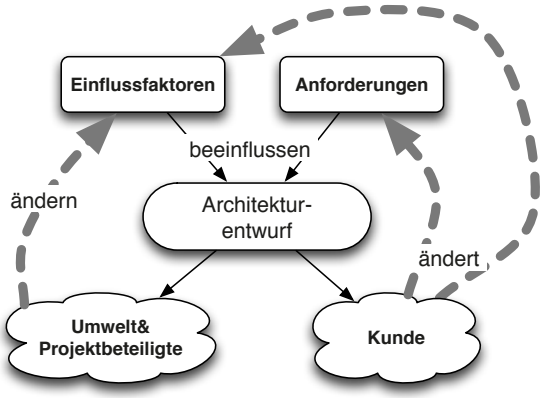

Bild 3.3
Iterativer Entwurf
aufgrund veränderter Anforderungen

Im vorigen Kapitel haben Sie unter dem Begriff „Architecture Business Cycle“ bereits eine andere Art der *Iteration* von Architekturen kennengelernt.

Treffen Sie Architektur- und Entwurfsentscheidungen bewusst, angemessen und systematisch

Machen Sie sich vor und während Ihrer Architekturentwicklung bewusst, welche architekturrelevanten Entscheidungen Sie oder andere Stakeholder treffen müssen. Stimmen Sie mit betroffenen Stakeholdern ab, ob und wann diese Entscheidung notwendig wird – es kann sowohl für schnelle wie auch für verzögerte Entscheidungen gute Gründe geben.

Entscheidungen dokumentieren

Wenn Sie eine Entscheidung für besonders wichtig oder bemerkenswert halten, dann sollten Sie diese dokumentieren: Halten Sie dazu den genauen Entscheidungsumfang („Was ist das Problem?“) sowie die Auslöser für diese Entscheidung („Warum müssen wir das entscheiden?“) fest sowie

die Rahmenbedingungen, Ihre getroffenen Annahmen, Risiken und mögliche (verworfene) Alternativen.[3] In jedem Fall sollten Sie Ihre Entscheidung begründen.

Entwickeln Sie Mut zu Entscheidungen! Akzeptieren Sie, dass Sie so manche unsichere Entscheidung treffen müssen (und erinnern Sie sich bisweilen an das Zitat von Philippe Kruchten aus Abschnitt 2.2 ...).

3.1 Informationen sammeln

Noch bevor Sie eigene Lösungsideen entwickeln, sollten Sie recherchieren, wie andere Architekten vor Ihnen ähnliche Probleme gelöst haben:

Quellen für Wiederverwendung finden

- Beginnen Sie mit Ihrer eigenen Erfahrung – Ihre Kunden erwarten sowohl Domänenwissen als auch technisches Wissen von Ihnen.
- Prüfen Sie die Projekte innerhalb Ihrer Organisation, ob sich Ergebnisse wiederverwenden lassen. Seien Sie dabei pragmatisch!
- Suchen Sie im Internet nach Beschreibungen ähnlicher Systeme. Vielleicht können Sie passende Komponenten dazukaufen.
- Suchen Sie in der technischen Literatur und durchforsten Sie die Entwurfsmuster auf Ihrem Gebiet.

In vielen Fällen bekommen Sie dadurch Beispiele und Muster und müssen nur noch kleine Teile völlig neu entwerfen.

Sie sollten ein tiefes Misstrauen gegenüber solchen Lösungen hegen, die sich in den Grundzügen nicht auf bekannte Dinge zurückführen lassen. Nur die wenigsten, die solche (angeblich innovativen) Lösungen präsentieren, sind wirklich geniale Erfinder – die meisten haben nur schlecht recherchiert.

[3] Stefan Zörner schlägt vor, bei wichtigen Entscheidungen auch die Namen der Entscheider festzuhalten.

3.2 Anforderungen klären

Als Grundlage Ihrer Entwurfstätigkeiten sollten Sie eine Vorstellung wichtiger Systemeigenschaften und -anforderungen besitzen und mindestens folgende Fragen über das System beantwortet haben:

- Was ist die Kernaufgabe des Systems (= funktionale Anforderungen)? Welches sind die wichtigsten Elemente der Fachdomäne?
- Um welche Kategorie von System handelt es sich?
- Welche Qualitätsanforderungen muss das System erfüllen (= Architekturziele)?
- Welche Personen oder Gruppen haben Interessen am System (= Stakeholder)?
- Was sind die (fachlichen und technischen) Nachbarsysteme (= externe Schnittstellen, Kontextabgrenzung)?

Die Antworten auf diese Fragen werden Ihnen (und dem Entwicklungsteam) eine Orientierung für erste Lösungsansätze geben.

3.2.1 Was ist die Kernaufgabe des Systems?

- Beschreiben Sie die Kernaufgabe und Verantwortlichkeit des Systems in zwei bis drei Sätzen. Formulieren Sie positiv und benutzen Sie die Kernbegriffe der Fachdomäne.
- Fügen Sie die wichtigsten Begriffe oder Aspekte der Fachdomäne hinzu; wenige Begriffe (fünf bis zehn) genügen häufig.
- Stimmen Sie diese Formulierung mit Kunden und Auftraggebern ab!

Beispiel: Das System unterstützt Call-Agents bei der Erfassung und Bearbeitung von Schadensmeldungen von Privatkunden.

Die wichtigsten Aspekte der Fachdomäne sind Vorfall, Schaden, Partner und Vertrag.

Eine solche kurze Formulierung erleichtert die Kommunikation über das System. Sie definiert für alle Beteiligten das wichtigste Ziel (= das System). Gleichzeitig schafft sie einen begrifflichen Kontext, an dem sich alle Beteiligten orientieren können.

Manchmal können Sie diese Aussagen aus der Anforderungsanalyse übernehmen. Andernfalls müssen Sie die Kernaufgabe mit Ihren Auftraggebern oder Kunden selbst formulieren.

3.2.2 Welche Kategorie von System?

Meist geben die Kernaufgaben eines Systems bereits Aufschluss, zu welcher der folgenden Kategorien es gehört:

- Interaktive Online-Systeme: Auch als operationale Systeme bezeichnet, arbeiten diese Systeme als Teil der normalen Geschäftsprozesse in Unternehmen. In den meisten Fällen enthalten sie Operationen auf Daten (Transaktionen, Einfüge-, Änderungs- und Löschoperationen), die vom Ablauf her in die Benutzeroberfläche eingebettet sind. Die Art der Transaktionen ist festgelegt. Die Systeme operieren auf möglichst aktuellen Datenbeständen. Sie erfordern ein hohes Maß an Systemverfügbarkeit und Performance.
- Mobile Systeme: Eine (moderne) Variante von Online-Systemen – durch die starke Verbreitung von Smartphones und Tablet-Computern sehr wichtig geworden. Starker Fokus auf Benutzeroberfläche und Benutzbarkeit, oftmals per Internet an Backend-Systeme angebunden.
- Entscheidungsunterstützungssysteme (*decision support system*): arbeiten oftmals auf Kopien der aktuellen Unternehmensdaten (*data warehouse*) und enthalten hauptsächlich lesende Datenzugriffe. Die Art der Anfragen an die Daten ist flexibel. Benutzer können neue Anfragen (*queries*) formulieren. Daher ist die Laufzeit von Anfragen im Vorfeld kaum abschätzbar. Solche Systeme tolerieren höhere Ausfallzeiten und geringere Performance.
- Hintergrundsysteme (Offline-Systeme, Batch-Systeme): dienen hauptsächlich der Datenmanipulation, oftmals zur Vor- oder Nachverarbeitung vorhandener Datenbestände; werden zur Interaktion mit anderen Systemen eingesetzt. In Kapitel 7 finden Sie mehr zum Thema Integration.
- Eingebettete Systeme (*embedded systems*): arbeiten eng verzahnt mit spezieller Hardware. Ein Beispiel für eingebettete Systeme sind Mobiltelefone (viele haben auch Echtzeitanforderungen).
- Systeme mit Echtzeitanforderungen (*real-time systems)*: Operationen werden innerhalb garantierter Zeiten fertiggestellt. Beispiel: Produktionssteuerung (Fließbänder mit festen Taktzeiten), Herzschrittmacher. Solche Systeme werde ich im Folgenden nicht mehr weiter ausführen – sie sind eine Wissenschaft für sich. ☹

Wenn das System zu mehreren der genannten Kategorien gehört, sollte die Architektur aus entsprechenden Teilsystemen bestehen, die jeweils zu einer Kategorie gehören.

3.2.3 Wesentliche Qualitätsanforderungen ermitteln

Qualität, laut Duden definiert als „Beschaffenheit, Güte, Wert“, stellt ein wichtiges Ziel für Softwarearchitekten dar. Bei Qualität handelt es sich allerdings um ein vielschichtiges Konzept, das mit einer Reihe gravierender Probleme behaftet ist:

- Qualität ist nur indirekt messbar: Es gibt kein absolutes Maß für die Qualität eines Produkts, höchstens für einzelne Eigenschaften (etwa: Zeit- oder Ressourceneffizienz).

- Qualität ist relativ: Verschiedene Stakeholder haben unterschiedliche Qualitätsbegriffe und -anforderungen.
  - Manager und Auftraggeber fordern oft Kosteneffizienz, Flexibilität und Wartbarkeit.
  - Endanwender fordern hohe Performance und einfache Benutzbarkeit.
  - Projektleiter fordern Parallelisierbarkeit der Implementierung und gute Testbarkeit.
  - Betreiber fordern Administrierbarkeit und Sicherheit.
- Qualität der Architektur korreliert nicht notwendigerweise mit der Qualität des Endprodukts: Gute Architekturen können schlecht implementiert sein und dadurch die Qualität des Gesamtsystems mindern. Aus hervorragendem Quellcode kann jedoch nicht auf die Qualität der Architektur geschlossen werden. Insgesamt gilt daher: Architektur ist für die Qualität eines Systems notwendig, aber nicht hinreichend.
- Die Erfüllung sämtlicher funktionaler Anforderungen lässt kaum Aussagen über die Erreichung der Qualitätsanforderungen zu. Betrachten Sie das Beispiel eines einfachen Sortierverfahrens: Die (triviale) Anforderung, eine Menge von Variablen gemäß einem vorgegebenen Sortierkriterium aufsteigend zu ordnen, ist eine beliebte Programmieraufgabe für Einsteiger. Nun denken Sie an einige zusätzliche nichtfunktionale Anforderungen, etwa:
  - Sortierung großer Datenmengen (Terabyte), die nicht mehr zeitgleich im Hauptspeicher gehalten werden können,
  - Sortierung robust gegenüber unterschiedlichen Sortierkriterien (Umlaute, akzentuierte Zeichen, Phoneme, Ähnlichkeitsmaße und anderes),
  - Sortierung für viele parallele Benutzer,
  - Sortierung unterbrechbar für lang laufende Sortiervorgänge,
  - Erweiterbarkeit um weitere Algorithmen, beispielsweise für ressourcenintensive Vergleichsoperationen,
  - Entwickelbarkeit im räumlich verteilten Team.

Diese Qualitätsanforderungen können von naiven Implementierungen nicht erfüllt werden – dazu bedarf es grundlegender architektonischer Maßnahmen!

Viele Publikationen[4] über Softwarearchitektur ignorieren das Thema der Qualitätsanforderungen völlig. Es scheint fast, als fielen Verständlichkeit, Wartbarkeit und Performance von Systemen als Nebenprodukte ab, wenn eifrige Entwickler nur in ausreichender Menge Design- und Architekturmuster anwenden[5]. Das Gegenteil ist der Fall:

Qualitätsanforderungen müssen explizite Entwurfsziele sein. Treffen Sie Entscheidungen zur Erreichung solcher Ziele bewusst und frühzeitig. Qualität muss explizit konstruiert werden – sie entsteht nicht von selbst!

[4] Asche auf mein Haupt – in der ersten Auflage dieses Buchs habe ich die gleiche Unterlassung begangen. Sie halten zum Glück eine neuere Auflage in der Hand. J

[5] Leider helfen selbst Ansätze wie „clean-code“ nur begrenzt: Auch lupenreiner „clean-code“ kann elementare Qualitätsanforderungen wie Datensicherheit, Benutzbarkeit oder Betreibbarkeit verletzen!

Typische Qualitätsanforderungen

Die Qualität von Softwaresystemen wird immer bezogen auf einzelne Eigenschaften oder Merkmale. Beispiele für solche Merkmale sind Effizienz (Performance), Verfügbarkeit, Änderbarkeit und Verständlichkeit.

Es gibt eine ganze Reihe unterschiedlicher Definitionen von Qualitätsmodellen und Qualitätsmerkmalen. Alle definieren einige zentrale Qualitätseigenschaften (beispielsweise Zuverlässigkeit, Effizienz, Wartbarkeit, Portabilität etc.) und verfeinern diese Eigenschaften durch eine Hierarchie weiterer Merkmale.

Egal, welches dieser Modelle Sie verwenden: Achten Sie innerhalb Ihrer Systeme oder Projekte auf eine einheitliche Terminologie für Qualitätsanforderungen.

Tabelle 3.1 zeigt Ihnen die Qualitätsmerkmale gemäß DIN/ISO 9126.[6] Diese Norm enthält die wesentlichen Begriffe rund um Softwarequalität.

Falls Sie in dieser Tabelle (vergeblich) nach dem Stichwort „Performance“ suchen – im DIN-normierten Sprachgebrauch heißt es „Effizienz“.

Tabelle 3.1 Qualitätsmerkmale nach DIN/ISO 9126

Funktionalität	Vorhandensein von Funktionen mit festgelegten Eigenschaften; diese Funktionen erfüllen die definierten Anforderungen.
▪ Angemessenheit	Liefern der richtigen oder vereinbarten Ergebnisse oder Wirkungen, z. B. die benötigte Genauigkeit von berechneten Werten
▪ Richtigkeit	Eignung der Funktionen für spezifizierte Aufgaben, z. B. aufgabenorientierte Zusammensetzung von Funktionen aus Teilfunktionen
▪ Interoperabilität	Fähigkeit, mit vorgegebenen Systemen zusammenzuwirken. Hierunter fällt auch die Einbettung in die Betriebsinfrastruktur.
▪ Ordnungsmäßigkeit	Erfüllung von anwendungsspezifischen Normen, Vereinbarungen, gesetzlichen Bestimmungen und ähnlichen Vorschriften
▪ Sicherheit	Fähigkeit, unberechtigten Zugriff, sowohl versehentlich als auch vorsätzlich, auf Programme und Daten zu verhindern
Zuverlässigkeit	Fähigkeit der Software, ihr Leistungsniveau unter festgelegten Bedingungen über einen festgelegten Zeitraum zu bewahren
▪ Reife	Geringe Versagenshäufigkeit durch Fehlzustände
▪ Fehlertoleranz	Fähigkeit, ein spezifiziertes Leistungsniveau bei Softwarefehlern oder Nichteinhaltung ihrer spezifizierten Schnittstelle zu bewahren
▪ Wiederherstellbarkeit	Fähigkeit, bei einem Versagen das Leistungsniveau wiederherzustellen und die direkt betroffenen Daten wiederzugewinnen

[6] Es gibt einige Nachfolgestandards (ISO 25000 – 25030), die jedoch in die Praxis bislang wenig Einzug gehalten haben.

Tabelle 3.1 (*Fortsetzung*) Qualitätsmerkmale nach DIN/ISO 9126

Benutzbarkeit	Aufwand, der zur Benutzung erforderlich ist, und individuelle Beurteilung der Benutzung durch eine festgelegte oder vorausgesetzte Benutzergruppe; hierunter fällt auch der Bereich Softwareergonomie.
▪ Verständlichkeit	Aufwand für den Benutzer, das Konzept und die Anwendung zu verstehen
▪ Erlernbarkeit	Aufwand für den Benutzer, die Anwendung zu erlernen (z. B. Bedienung, Ein-, Ausgabe)
▪ Bedienbarkeit	Aufwand für den Benutzer, die Anwendung zu bedienen
Effizienz	Verhältnis zwischen dem Leistungsniveau der Software und dem Umfang der eingesetzten Betriebsmittel unter festgelegten Bedingungen
▪ Zeitverhalten	Antwort- und Verarbeitungszeiten sowie Durchsatz bei der Funktionsausführung
▪ Verbrauchsverhalten	Anzahl und Dauer der benötigten Betriebsmittel für die Erfüllung der Funktionen
Änderbarkeit	Aufwand, der zur Durchführung vorgegebener Änderungen notwendig ist Änderungen: Korrekturen, Verbesserungen oder Anpassungen an Änderungen der Umgebung, der Anforderungen und der funktionalen Spezifikationen
▪ Analysierbarkeit	Aufwand, um Mängel oder Ursachen von Versagen zu diagnostizieren oder um änderungsbedürftige Teile zu bestimmen
▪ Modifizierbarkeit	Aufwand zur Ausführung von Verbesserungen, zur Fehlerbeseitigung oder Anpassung an Umgebungsänderungen
▪ Stabilität	Wahrscheinlichkeit des Auftretens unerwarteter Wirkungen von Änderungen
▪ Prüfbarkeit	Aufwand, der zur Prüfung der geänderten Software notwendig ist
Übertragbarkeit	Eignung der Software, von einer Umgebung in eine andere übertragen zu werden. Umgebung kann organisatorische Umgebung, Hardware- oder Softwareumgebung einschließen.
▪ Anpassbarkeit	Software an verschiedene festgelegte Umgebungen anpassen
▪ Installierbarkeit	Aufwand, der zum Installieren der Software in einer festgelegten Umgebung notwendig ist
▪ Konformität	Grad, in dem die Software Normen oder Vereinbarungen zur Übertragbarkeit erfüllt
▪ Austauschbarkeit	Möglichkeit, diese Software anstelle einer spezifizierten anderen in der Umgebung jener Software zu verwenden, sowie der dafür notwendige Aufwand

Qualitätsanforderungen über Szenarien konkretisieren

Jetzt kennen Sie zwar die wichtigsten Qualitätsmerkmale, Sie benötigen aber noch ein Mittel, um diese Merkmale praxisgerecht für Ihre Projekte zu konkretisieren und zu definieren. Hierzu eignen sich Szenarien (nach [Bass+03]). Szenarien beschreiben, was beim Eintreffen eines Stimulus auf ein System in bestimmten Situationen geschieht. Sie charakterisieren damit das Zusammenspiel von Stakeholdern mit dem System. Szenarien konkretisieren Qualitätsanforderungen.

Ich möchte Ihnen die Anwendung von Szenarien zur Konkretisierung von Qualitätsanforderungen anhand einiger Beispiele[7] verdeutlichen.

- *Anwendungsszenarien (beschreiben das Verhalten des Systems bei seiner Nutzung):*
  - Die Antwort auf eine Angebotsanfrage muss Endbenutzern im Regelbetrieb in weniger als fünf Sekunden dargestellt werden. Im Betrieb unter Hochlast (Jahresendgeschäft) darf eine Antwort bis zu 15 Sekunden dauern; in diesem Fall ist vorher ein entsprechender Hinweis darzustellen.
  - Ein Benutzer ohne Vorkenntnisse muss bei der erstmaligen Verwendung des Systems innerhalb von 15 Minuten in der Lage sein, die gewünschte Funktionalität zu lokalisieren und zu verwenden.
  - Bei Eingabe illegaler oder fehlerhafter Daten in die Eingabefelder muss das System entsprechende spezifische Hinweistexte ausgeben und anschließend im Normalbetrieb weiterarbeiten.
- *Änderungsszenarien (beschreiben gewünschte Reaktionen bei Änderungen am System oder der Umgebung):*
  - Die Programmierung neuer Versicherungstarife muss in weniger als 30 Personentagen möglich sein.
  - Die Unterstützung einer neuen Browser- oder Client-JDK-Version muss in weniger als 30 Personentagen programmiert und getestet werden können.
  - Bei Ausfall einer CPU muss das Ersatzsystem (Hot-Spare) im Normalbetrieb innerhalb von 15 Minuten online sein.
  - Die Anbindung eines neuen CRM-Systems[8] muss innerhalb von 60 Tagen möglich sein.

Bestandteile von Szenarien

Nach diesen Beispielen können Sie sicherlich etwas Methodik vertragen: Szenarien bestehen aus folgenden wesentlichen Teilen (in Klammern die Terminologie aus [Bass+03]):

- Auslöser (*stimulus*): beschreibt eine spezifische Zusammenarbeit des (auslösenden) Stakeholder mit dem System. Beispiele: Ein Benutzer ruft eine Funktion auf, ein Entwickler programmiert eine Erweiterung, ein Administrator installiert oder konfiguriert das System.
- Quelle des Auslösers (*source*): beschreibt, woher der Auslöser stammt. Beispiele: intern oder extern, Benutzer, Betreiber, Angreifer, Manager.

[7] Zahlreiche Beispiele von Szenarien für Qualitätsanforderungen finden Sie unter *https://github.com/arc42/quality-requirements*

[8] CRM (Customer-Relationship-Management, deutsch *Verwaltung von Kundenbeziehungen*) hat das Ziel, Kundenbeziehungen in einem Unternehmen zu organisieren und somit die Kundenzufriedenheit und -bindung sowie die Profitabilität zu erhöhen (nach: [Wikipedia]).

- Umgebung (*environment*): beschreibt den Zustand des Systems zum Zeitpunkt des Auslösers. Befindet sich das System unter Normal- oder Höchstlast? Ist die Datenbank verfügbar oder nicht? Sind Benutzer online oder nicht? Hier sollten Sie alle Bedingungen beschreiben, die für das Verständnis des Szenarios wichtig sind.
- Systembestandteil (*artifact*): beschreibt, welcher Bestandteil des Systems vom Auslöser betroffen ist. Beispiele: Gesamtsystem, Datenbank, Webserver.
- Antwort (*response*): beschreibt, wie das System durch seine Architektur auf den Auslöser reagiert. Wird die vom Benutzer aufgerufene Funktion ausgeführt? Wie lange benötigt der Entwickler zur Programmierung? Welche Systemteile sind von Installation/Konfiguration betroffen?
- Antwortmetrik (*response measure*): beschreibt, wie die Antwort gemessen oder bewertet werden kann. Beispiele: Ausfallzeit in Stunden, Korrektheit Ja/Nein, Änderungszeit in Personentagen, Reaktionszeit in Sekunden.

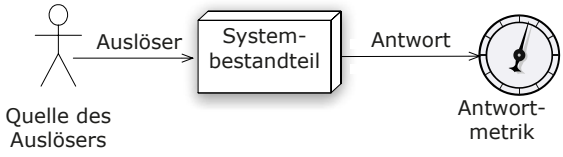

Bild 3.4
Schematische Darstellung von Szenarien (nach [Bass+03])

Qualitätsanforderungen explizit über Szenarien zu beschreiben, ist einfach und alleine deswegen nützlich!

Zusätzlich dienen diese Szenarien als wesentliche Grundlage der systematischen Architekturbewertung, die Sie in Kapitel 8 näher kennenlernen.

3.2.4 Relevante Stakeholder ermitteln

Stakeholder

Welche Rollen oder Personen (= Stakeholder) innerhalb oder außerhalb der Organisation haben ein Interesse am System? Beispiele sind Benutzer der Kernfunktionalität („Anwender"), Administratoren und Betreiber, Benutzer mit Sonderfunktionen (Genehmiger, Prüfer, Auditoren oder Ähnliche).

Welche Stakeholder stehen dem (neuen) System negativ gegenüber? Die jeweilige Einstellung der Stakeholder prägt die Art der Informationen, die sie Ihnen über das System geben werden.[9]

Insbesondere müssen Sie wissen, welche Erwartungshaltung die Stakeholder gegenüber dem Architektur- und Entwicklungsteam haben, welche Artefakte oder Dokumente Sie liefern und pflegen müssen.

Ich empfehle Ihnen, die Stakeholder Ihres Systems als Tabelle in die Architekturdokumentation aufzunehmen – wobei Sie folgende Informationen festhalten sollten:

[9] Softwaresysteme sind soziale Systeme. Als Architekt werden Sie daher häufig mit subjektiven Eindrücken statt Tatsachen konfrontiert, mit Meinungen statt Fakten, mit unrealistischen Erwartungen statt begründeter Anforderungen. Stellen Sie sich darauf ein!

Tabelle 3.2 Aufbau der Stakeholder-Tabelle

Rolle	Welche Aufgabe/Verantwortung tragen diese Stakeholder bezüglich des Systems?
Beschreibung	
Ziel/Intention	Welche konkreten Ziele verfolgen die Stakeholder bezüglich des Systems?
Erwartungshaltung	Welche konkreten Ergebnisse, Dokumente, Artefakte erwarten diese Stakeholder von Ihnen, der Architektur oder dem Entwicklungsteam?
Kontakt*	Sie benötigen für viele Ihrer Aufgaben Feedback der Stakeholder, daher sollten Sie deren Kontaktinformationen (E-Mail, Telefon) kennen.
Relevanz für Abnahme	Können diese Stakeholder die Abnahme oder den Einsatz des Systems erlauben, fördern oder behindern?

* Das mag formalistisch klingen, ist im Projektalltag jedoch praktisch zu wissen!

Beispiele für Stakeholder

Nachfolgend nenne ich Ihnen einige Beispiele für Stakeholder, die in meinen eigenen Projekten relevant waren (nicht alle auf einmal ...):

Management, Auftraggeber, Projektsteuerkreis, sonstige Projektgremien, PMO, Projektmanager, Produktmanager, Fachbereich, Unternehmens-/Enterprise-Architekt, Architekturabteilung, Methodenabteilung, QS-Abteilung, IT-Strategie, (interner) Softwarearchitekt, Softwaredesigner, Entwickler, Tester, Konfigurationsmanager, Build-Manager, Release-Manager, Wartungsteam, externe Dienstleister, Zulieferer, Hardwaredesigner, Rollout-Manager, Infrastrukturplaner, Sicherheitsbeauftragter, Behörde, Aufsichtsgremium, Auditor, Mitbewerber/Konkurrent, Endanwender, Fachpresse, Fachadministrator, Systemadministrator, Operator, Hotline, Betriebsrat, Lieferant von Standardsoftware, verbundene Projekte, Normierungsgremium, Gesetzgeber ...

3.2.5 Fachlichen und technischen Kontext ermitteln

Sie sollten immer eine präzise Vorstellung der (fachlichen wie technischen) Umgebung eines Systems besitzen (= Kontext), d. h. welche Nachbarsysteme oder Benutzergruppen mit dem System interagieren.

Klären Sie möglichst frühzeitig:

- Welche Schnittstellen gibt es zu externen Systemen? Hierzu zählen:
  - Schnittstellen anderer genutzter Systeme,
  - Schnittstellen, die für andere Systeme bereitgestellt werden.
- Sind diese externen Schnittstellen stabil und zuverlässig oder volatil? Das bedeutet: Sind die Fremdsysteme bezüglich ihrer Schnittstellen stabil?
- Wie fehlertolerant muss das neue System gegenüber den fremden Systemen sein?

- Handelt es sich um funktionale Schnittstellen oder Datenschnittstellen? Können Sie über diese Schnittstellen Funktionen oder Methoden des anderen Systems aufrufen oder funktioniert die Schnittstelle nur über den Austausch von Daten?
- Kommuniziert das System über die Schnittstellen synchron oder asynchron?
- Wie performant sind die anderen Systeme? Wie beeinflusst das Verhalten der Schnittstellen die Performance des Gesamtsystems?
- Gibt es die Möglichkeit, die Schnittstelle auf Seiten des anderen Systems zu ändern? Liegt der Quellcode der Schnittstelle vor?

- Dokumentieren Sie, welche Schnittstellen Sie bis jetzt identifiziert haben.
- Wenn das System Schnittstellen zu Fremdsystemen besitzt, sollten Sie über einen funktionalen Prototyp die technische Machbarkeit „end-to-end“ verifizieren. Insbesondere kritische oder riskante Schnittstellen sollten Sie möglichst frühzeitig auf diese Weise überprüfen.

Kontext als Diagramm – Erläuterung als Tabelle

Ich rate Ihnen, den Kontext Ihres Systems grafisch zu dokumentieren – und dieses Diagramm dann mit einer Tabelle zu erläutern (Bild 3.5).

Tabellarisch erklären Sie dazu die genauen Bezeichnungen aller Schnittstellen, die Richtung (ob sie aus der Sicht Ihres Systems benötigte (*required*) oder angebotene (*provided*) Schnittstellen sind) und die fachliche Bedeutung in Stichworten.

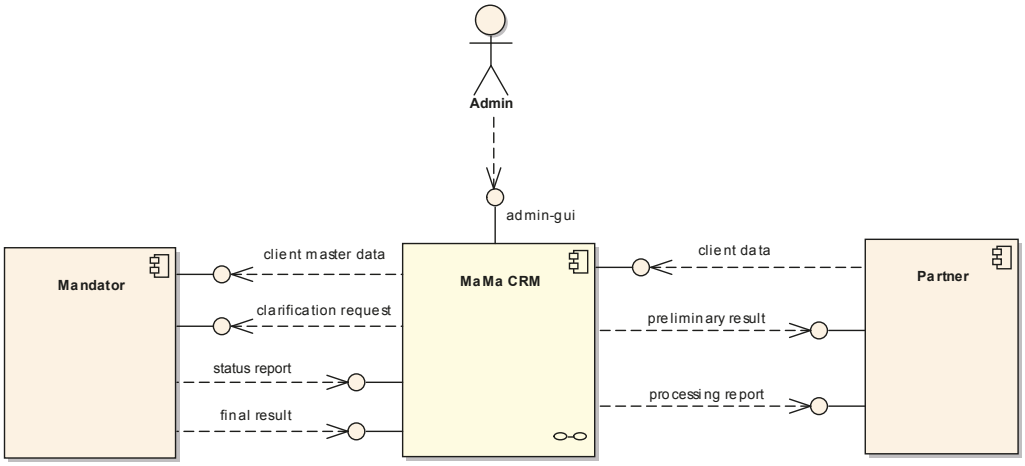

Bild 3.5 Beispiel einer Kontextabgrenzung

3.3 Einflussfaktoren und Randbedingungen ermitteln

Um als Softwarearchitekt anwendungs- und problembezogene Entwurfsentscheidungen zu treffen, müssen Sie die Faktoren kennen, die Ihre Architekturen beeinflussen oder einschränken werden (siehe dazu Bild 3.6). Diese Faktoren schränken Sie in Ihren Entscheidungen ein und besitzen daher einen prägenden Einfluss auf das Gesamtsystem.

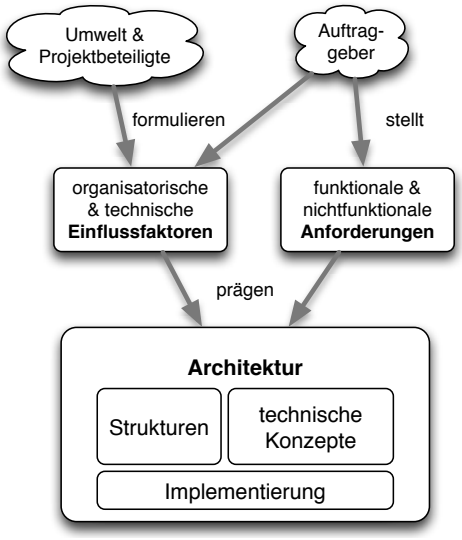

Bild 3.6
Anforderungen und Einflussfaktoren prägen den Architekturentwurf.

Aus den Systemanforderungen sowie den Anforderungen der Systemumwelt ermitteln Sie architekturrelevante Einflussfaktoren. Im Folgenden stelle ich Ihnen einige typische Einflussfaktoren vor.

Randbedingungen und Einflussfaktoren fallen in folgende Kategorien:

- Organisatorische und politische Faktoren. Manche solcher Faktoren wirken auf ein bestimmtes System ein. Andere beeinflussen sämtliche Projekte innerhalb einer Organisation. [Rechtin2000] charakterisiert diese Faktoren als *facts of life*.
- Technische Faktoren. Sie prägen das technische Umfeld des Systems und seiner Entwicklung.

Meiner Erfahrung nach tendieren Softwarearchitekten dazu, die politischen und organisatorischen Faktoren zu unterschätzen und zu vernachlässigen. Das kann im Extremfall dazu führen, dass an sich lauffähige Systeme nicht zum Einsatz kommen.

Das hat mehrere Gründe: Erstens bestimmen die technischen Faktoren ein System viel offensichtlicher. Zweitens sind Softwarearchitekten in technischen Themen oftmals erfahrener als in organisatorischen oder gar politischen. Drittens herrscht in Softwareprojekten manchmal ein technischer Zweckoptimismus, getreu dem Motto: Mit der neuen X-Technik und dem Y-Werkzeug haben wir unsere Probleme im Griff.

Wie [Rechtin2000] es treffend formuliert: *Es gibt keine rein technischen Probleme*. Sie werden schnell zu organisatorischen oder politischen Schwierigkeiten und damit entgleiten sie auch der Kontrolle rein technisch orientierter Projektbeteiligter.

Die Einflussfaktoren und Randbindungen auf eine konkrete Softwarearchitektur sollten Sie in einer Tabelle (als Bestandteil der Architekturdokumentation) festhalten.

Organisatorische Faktoren

Diese Faktoren beziehen sich im weitesten Sinne auf die Umgebung, in der das System erstellt wird. Sie prägen das System indirekt. Hierzu zählen Aspekte wie Termin- und Budgetplanung, vorhandene Mitarbeiter, technische Ressourcen, Entwicklungsprozesse, Vorgaben bezüglich Werkzeuge und Ähnliches. Tabelle 3.3 gibt einen Überblick über typische organisatorische Faktoren.

Einige Tipps zur Identifikation von organisatorischen Faktoren und Risiken Ihrer Projekte:

- Denken Sie negativ. Murphys Regel besagt: Wenn etwas schiefgehen kann, wird es irgendwann schiefgehen. Und es geschieht immer im denkbar schlechtesten Moment.
- Politik und Organisation, nicht Technik, setzen die Grenzen dafür, was ein Projekt erreichen kann und darf. Bringen Sie die „politischen" Stakeholder auf Ihre Seite. Stellen Sie sicher, dass Sie die Intention der „Politiker" innerhalb Ihrer Organisation richtig verstanden haben.
- Eine fundamentale Gleichung lautet: „Geld = Politik" ([Rechtin2000]). Die Politik gibt Projekten die Kostenregeln vor. Diese Kostenregeln besitzen prägenden Einfluss auf Architekturen.
- Finden Sie heraus, welche Ziele die einzelnen Stakeholder mit dem System verfolgen. Oftmals werden in einem Projekt verschiedene Ziele verfolgt!
- Die besten technischen Lösungen sind nicht unbedingt die besten politischen Lösungen. Im Regelfall sind „politische" Stakeholder die Auftraggeber und Eigentümer von Projekten (nach [Rechtin2000]). Sie entscheiden meist nach anderen Kriterien als „technische" Stakeholder.
- Betrachten Sie systemübergreifende Prozesse oder Rollen innerhalb der Organisation. Beispiel: In vielen großen Unternehmen gibt es Vorgaben zur Datenmodellierung und zum Datenbankdesign (die von dedizierten Datenbankadministratoren geprüft und freigegeben werden müssen).
- Beziehen Sie andere Projekte innerhalb der Organisation in Ihre Betrachtung mit ein. Sie können aus deren Verlauf viel über Stärken und Schwächen der Organisation hinsichtlich Softwareerstellung lernen.
- Die Erfahrung der beteiligten Entscheider spielt meistens eine wichtige Rolle. So schränkt eine negative Erfahrung des Auftraggebers mit einer bestimmten Technologie Ihre Entwurfsalternativen möglicherweise ein.[10]
- Werfen Sie einen Blick auf das Risikomanagement Ihrer Projektleitung. Eventuell finden Sie dort neue und für die Architektur wichtige Faktoren.

[10] Edward Yourdon rät in seinem Buch „Death March: Surviving Mission Impossible Projects" dazu, bei einer nicht akzeptablen Fülle solcher Einschränkungen das Projekt zu verlassen oder zu kündigen, um das eigene (berufliche) Überleben zu sichern.

Tabelle 3.3 Typische organisatorische Einflussfaktoren

Faktor	Erläuterung
Organisation und Struktur	
Organisationsstruktur beim Auftraggeber	▪ Droht Änderung von Verantwortlichkeiten? ▪ Änderung von Ansprechpartnern
Organisationsstruktur des Projektteams	▪ Mit/ohne Unterauftragnehmer ▪ Entscheidungsbefugnis der Projektleiterin
Entscheidungsträger	▪ Erfahrung mit ähnlichen Projekten ▪ Risiko-/Innovationsfreude
Bestehende Partnerschaften oder Kooperationen	▪ Hat die Organisation bestehende Kooperationen mit bestimmten Softwareherstellern? ▪ Solche Partnerschaften geben oftmals (unabhängig von Systemanforderungen) Produktentscheidungen vor.
Eigenentwicklung oder externe Vergabe	▪ Selbst entwickeln oder an externe Dienstleister vergeben?
Entwicklung als Produkt oder zur eigenen Nutzung?	Bedingt andere Prozesse bei Anforderungsanalyse und Entscheidungen. Im Fall der Produktentwicklung: ▪ Neues Produkt für neuen Markt? ▪ Verbessertes Produkt für bestehenden Markt? ▪ Vermarktung eines bestehenden (eigenen) Systems ▪ Entwicklung ausschließlich zur eigenen Nutzung?
Ressourcen (Budget, Zeit, Personal)	
Festpreis oder Zeit/Aufwand?	Festpreisprojekt oder Abrechnung nach Zeit und Aufwand?
Zeitplan	Wie flexibel ist der Zeitplan? Gibt es einen festen Endtermin? Welche Stakeholder bestimmen den Endtermin?
Zeitplan und Funktionsumfang*	Was ist höher priorisiert: der Termin oder der Funktionsumfang?
Release-Plan	Zu welchen Zeitpunkten soll welcher Funktionsumfang in Releases/Versionen zur Verfügung stehen?
Projektbudget	Fest oder variabel? In welcher Höhe verfügbar?
Budget für technische Ressourcen	Kauf oder Miete von Entwicklungswerkzeugen (Hardware und Software)?
Team	Anzahl der Mitarbeiter und deren Qualifikation, Motivation und kontinuierliche Verfügbarkeit

Tabelle 3.3 (*Fortsetzung*) Typische organisatorische Einflussfaktoren

Faktor	Erläuterung
Organisatorische Standards	
Vorgehensmodell	Vorgaben bezüglich Vorgehensmodell? Hierzu gehören auch interne Standards zu Modellierung, Dokumentation und Implementierung.
Qualitätsstandards	Fällt die Organisation oder das System in den Geltungsbereich von Qualitätsnormen (wie ISO 9000)?
Entwicklungs-werkzeuge	Vorgaben bezüglich der Entwicklungswerkzeuge (etwa: CASE-Tool, Datenbank, Integrierte Entwicklungsumgebung, Kommunikations-software, Middleware, Transaktionsmonitor)
Konfigurations- und Versionsverwaltung	Vorgaben bezüglich Prozessen und Werkzeugen
Testwerkzeuge und -prozesse	Vorgaben bezüglich Prozessen und Werkzeugen
Abnahme- und Freigabeprozesse	▪ Datenmodellierung und Datenbankdesign ▪ Benutzeroberflächen ▪ Geschäftsprozesse (Workflow) ▪ Nutzung externer Systeme (etwa: schreibender Zugriff bei externen Datenbanken)
Service Level Agreements	Gibt es Vorgaben oder Standards hinsichtlich Verfügbarkeiten oder einzuhaltender Service-Levels?
Juristische Faktoren	
Haftungsfragen	▪ Hat die Nutzung oder der Betrieb des Systems mögliche rechtliche Konsequenzen? ▪ Kann das System Auswirkungen auf Menschenleben oder Gesundheit haben? ▪ Kann das System Auswirkungen auf Funktionsfähigkeit externer Systeme oder Unternehmen haben?
Datenschutz	Speichert oder bearbeitet das System „schutzwürdige“ Daten?
Nachweispflichten	Bestehen für bestimmte Systemaspekte juristische Nachweis-pflichten?
Internationale Rechtsfragen	▪ Wird das System international eingesetzt? ▪ Gelten in anderen Ländern eventuell andere juristische Rahmen-bedingungen für den Einsatz (Beispiel: Nutzung von Verschlüsse-lungsverfahren)?

* Tom DeMarco beschreibt die Probleme mit Zeitplan und Funktionsumfang in seinem amüsanten und lehrreichen Roman über Projektmanagement („Der Termin“, 1998 bei Hanser erschienen).

Zu den organisatorischen Einflussfaktoren ein kleines Beispiel:

Beispiel: Die Informatik-Tochterfirma eines Konzerns erhielt den internen Auftrag, ein internetbasiertes Informationssystem für Finanznachrichten und Geschäftsberichte zu entwickeln und zu vermarkten.

Das Entwicklungsteam schien anfangs frei von organisatorischen und technischen Einflüssen zu sein. Im Laufe der Entwicklung stellte sich heraus, dass die Firma mit Marktpartnern und Kunden projektübergreifende „Service Level Agreements" (SLAs) bezüglich der Verfügbarkeit von Software vertraglich vereinbart hatte. Als Konsequenz für die Architektur ergab sich die Notwendigkeit, eine (anfänglich nicht geplante) Überwachungskomponente zum verbindlichen Nachweis der Systemverfügbarkeit zu entwickeln. Dies machte eine kosten- und zeitintensive Änderung des gesamten Persistenzkonzepts notwendig.

Der zuständigen Projektleiterin gelang es, mit den betroffenen Endkunden des Systems eine Änderung der Nachweispflicht und der SLAs zu vereinbaren und damit zumindest den Termindruck der notwendigen Änderungen zu entschärfen. ■

Technische Faktoren

Technische Faktoren mit Relevanz für die Softwarearchitektur betreffen einerseits die technische Infrastruktur, also die Ablaufumgebung des Systems. Andererseits umfassen sie auch technische Vorgaben für die Entwicklung, einzusetzende Fremdsoftware und vorhandene Systeme.

Einige Tipps bei der Suche nach technischen Faktoren:

- Analysieren Sie andere Projekte innerhalb Ihrer Organisation. Befragen Sie Architekten und Projektleiter solcher Projekte.
- Betrachten Sie andere Systeme innerhalb der Organisation. Wie sieht die technische Umgebung dieser Systeme aus? Wie werden diese Systeme betrieben?
- Betrachten Sie die vorhandene Infrastruktur hinsichtlich Hardware und Software.
- Das Qualitätsmanagement der Organisation kann Hinweise auf weitere Einflussfaktoren geben.
- Gibt es Methoden, Standards oder Vorlagen für Softwareprojekte?
- Analog zum Tipp bei den organisatorischen Faktoren: Betrachten Sie systemübergreifende Abnahme- und Freigabeprozesse. Sie machen häufig Vorgaben zur Gestaltung von Datenmodellen, Datenbanken, Benutzeroberflächen, Geschäftsprozessen (Workflows), Sicherheit, Laufzeit- und Wartungsprozessen sowie der technischen Infrastruktur.

Tabelle 3.4 zeigt einige typische technische Einflussfaktoren.

Tabelle 3.4 Typische technische Einflussfaktoren

Faktor	Erläuterung
Hardwareinfrastruktur	Prozessoren, Speicher, Netzwerke, Firewalls und andere relevante Elemente der Hardwareinfrastruktur
Softwareinfrastruktur	Betriebssysteme, Datenbanksysteme, Middleware, Kommunikationssysteme, Transaktionsmonitor, Webserver, Verzeichnisdienste
Systembetrieb	Batch- oder Online-Betrieb des Systems oder notwendiger externer Systeme?
Verfügbarkeit der Laufzeitumgebung	Rechenzentrum mit 7 x 24 h Betriebszeit? Gibt es Wartungs- oder Backupzeiten mit eingeschränkter Verfügbarkeit des Systems oder wichtiger Systemteile?
Grafische Oberfläche	Existieren Vorgaben hinsichtlich grafischer Oberfläche (*Style Guide*)?
Bibliotheken, Frameworks und Komponenten	Sollen bestimmte „Softwarefertigteile“ eingesetzt werden?
Programmiersprachen	Objektorientierte, strukturierte, deklarative oder Regelsprachen, kompilierte oder interpretierte Sprachen?
Referenzarchitekturen	Gibt es in der Organisation vergleichbare oder übertragbare Referenzprojekte?
Analyse- und Entwurfsmethoden	Objektorientierte oder strukturierte Methoden?
Datenstrukturen	Vorgaben für bestimmte Datenstrukturen, Schnittstellen zu bestehenden Datenbanken oder Dateien
Programmierschnittstellen	Schnittstellen zu bestehenden Programmen
Programmiervorgaben	Programmierkonventionen, fester Programmaufbau
Technische Kommunikation	Synchron oder asynchron, Protokolle

Das alles sollte die Systemanalyse geliefert haben

In einer idealen Situation hat die Systemanalyse, das Requirements Engineering oder Ihr Product-Owner bereits alle bis hier aufgeführten Informationen ermittelt und passend aufbereitet. Gute Systemanalytiker entwickeln ein fachliches Modell, stellen Anforderungen, Einflussfaktoren und Randbedingungen systematisch dar und beschreiben die geforderten Qualitäten des Systems in Form von Szenarien.

Nach meiner Erfahrung betreiben jedoch nur wenige Projekte eine so ausführliche und methodische Systemanalyse. In den übrigen Projekten müssen die Software- und Systemarchitekten Teile dieser Analyse nachholen, insbesondere die hier dargestellten Einflussfaktoren, Randbedingungen, Risiken und Qualitätsmerkmale ermitteln und beschreiben.

3.4 Entwerfen und kommunizieren

Bis hierhin haben Sie die Voraussetzungen geschaffen, um angemessen entwerfen (und implementieren) zu können. Fast der gesamte restliche Teil des Buchs ist der Erklärung und Vertiefung dieser Aufgabe gewidmet (siehe Bild 3.7).

- Strukturen entwerfen: Hier geht es primär um den Aufbau des Systems aus Quellcode, die Struktur der Implementierung. Kapitel 4 hilft Ihnen weiter.
- Konzepte entwerfen: Hier geht es um die übergreifenden, meist technischen Belange. Kapitel 7 kümmert sich ausführlich darum.
- Sie sollten als verantwortungsbewusster Softwarearchitekt die Umsetzung (d. h. Implementierung und Test) des Systems im Detail begleiten. Dazu finden Sie einige Tipps am Ende dieses Abschnitts.
- Meistens arbeiten wir in Teams – dazu müssen Sie über Architektur, Entwürfe, Entscheidungen, Strukturen und Konzepte miteinander kommunizieren. Kapitel 5 gibt Ihnen hierzu einige Hinweise.
- Explizit zu bewerten, ob ein System und dessen Architektur die geltenden Anforderungen erfüllen können, gehört ebenfalls zu Ihren Aufgaben. Kapitel 8 stellt einen systematischen Ansatz dazu vor.

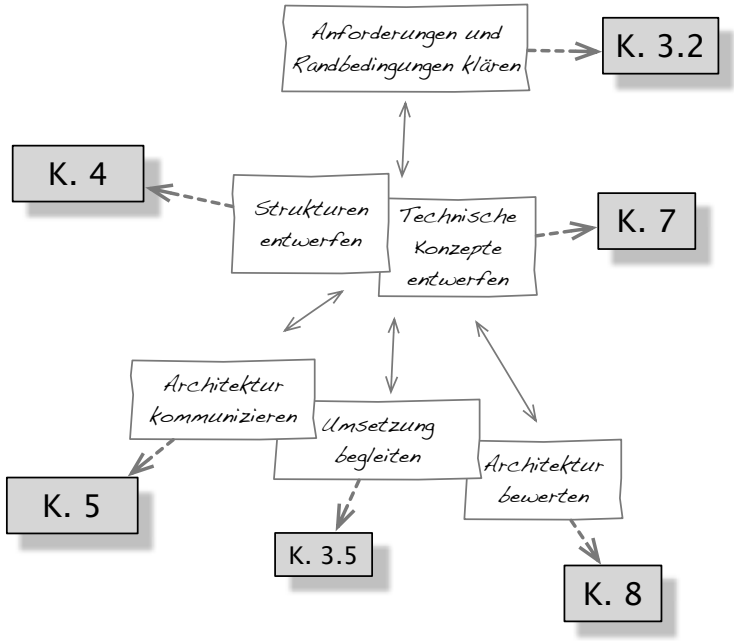

Bild 3.7 Hier finden Sie die weiteren Architektenaufgaben erklärt.

3.5 Umsetzung begleiten

Goldstücke suchen

Oftmals haben in meinen Projekten einzelne Entwickler hervorragende Verbesserungen gefunden und umgesetzt. Ihre Lösungen waren besser, einfacher, schneller oder günstiger als diejenigen des übrigen Teams oder der Architekturgruppe. Solche Lösungsansätze sollten Sie in jedem Fall in die Gesamtarchitektur übernehmen.[11] In anderen Industriezweigen nennt man dieses dauernde Suchen nach Verbesserungsmöglichkeiten „Kai-Zen" – ich nenne es gerne „Goldstücke suchen".

Bei der Entwicklung eines (Java-)Systems mit hohen Anforderungen an Flexibilität (genauer: Flexibilität von Datenim- und -exporten) hatten wir diese Herausforderung durch (XML-basierte) Konfiguration gelöst. Das hat gut funktioniert – allerdings war die Erstellung von Testfällen aufgrund der abstrusen Zahl spitzer Klammern eine wahre Strafarbeit für Entwickler und Tester …

Knapp ein Dutzend Entwickler fand über lange Zeit täglich neue Gründe, keine Testfälle für diese Konfiguration zu schreiben. Die Menge unserer Testfälle blieb sträflich gering …

Eines Tages hatte ein einzelner Entwickler plötzlich eine deutlich höhere Testrate. Auf Nachfragen stellte sich heraus, dass er unsere (bisher manuell erstellte) XML-Konfiguration sehr geschickt durch ein Sprachkonstrukt von Groovy* ergänzt – und damit den für eine komplexe Konfiguration nötigen Aufwand von bisher über 100 Zeilen XML-Konfiguration auf zehn bis 20 Zeilen Groovy-Quellcode gesenkt hatte. Und das auch noch mit großartiger IDE-Unterstützung.

Ein echtes „Goldstück" für das gesamte Team, das diese Idee begeistert aufgenommen hat – und damit ein wesentliches Qualitätsziel des Systems deutlich verbessern konnte.

* Groovy ist eine dynamische Sprache auf der JVM und hier ging es um die Groovy-MarkupBuilder (*http://groovy-lang.org*).

Missverständnisse aufdecken

Auf der anderen Seite könnten Entwickler auch Entwurfsentscheidungen oder Konzepte missverstehen oder schlecht umsetzen. In solchen Fällen sollten Sie als Architekt coachen und den Betroffenen die Entscheidungen oder Konzepte erläutern oder sie bei deren Implementierung unterstützen.

Quellcode lesen

In beiden Fällen müssen Sie in den Quellcode Ihrer Systeme schauen (= Code-Reviews) und die Ist-Implementierung mit dem gewünschten Soll vergleichen. Erst durch diese (Detail-) Arbeit können Sie wirklich erkennen, ob und wie Architekturentscheidungen letztlich umgesetzt werden.

[11] Eine Prüfung vorausgesetzt sowie die Eignung für die jeweilige Problemstellung.

Peer-Reviews und Delegation

Gerade in größeren Teams kann eine einzelne Person nicht mehr den gesamten erstellten oder geänderten Quellcode überprüfen. Codeanalyse-Werkzeuge können helfen, aber in wesentlichen Fällen (kritische oder wichtige Stellen im System) sollten Menschen solche Reviews durchführen (nur sie können nach Gründen fragen und diese situativ einschätzen).

Oft genügt es auch, kleine Teile des Quellcodes zu inspizieren, um den Aufwand dieser Aufgabe in angemessenem Rahmen zu halten.

3.6 Lösungsstrategien entwickeln

Nach der Klärung von Anforderungen, Einflussfaktoren und Risiken, insbesondere der geforderten Qualitätsmerkmale und Projektrisiken, kennen Sie also die Schwierigkeiten, die beim Entwurf des Systems auf Sie zukommen. Nun gilt es, passende Lösungsstrategien und -alternativen zu entwickeln.

Diese Strategien entwickeln Sie parallel zum Entwurf von Sichten und Konzepten (zu denen Sie in den folgenden Kapiteln noch viel mehr lesen).

Ich möchte mich in diesem Abschnitt auf Ratschläge und Anregungen zu organisatorischen Problemen beschränken, primär den Mangel an Zeit, Budget oder Erfahrung.

In Abschnitt 4.3.3 zeige ich Ihnen den methodischen Ansatz zur Erreichung von Qualitätsanforderungen (Quality-Driven Software Architecture)s und gebe dort Beispiele zu den Themen Performance, Flexibilität und Erweiterbarkeit.

Sollten einige Ihrer spezifischen Probleme in eher technischen Bereichen angesiedelt sein, kann Ihnen sicherlich der Katalog typischer Konzepte (Kapitel 7) weiterhelfen.

Strategien gegen organisatorische Risiken

Meiner Erfahrung nach leiden die meisten Softwareprojekte unter mindestens einem der folgenden (organisatorischen) Risiken:

- *Zu wenig Zeit:* Die Zeit bis zum Endtermin ist zu knapp bemessen, um die Anforderungen mit dem zugehörigen Projektteam zu erfüllen.
- *Zu wenig Budget:* Es mangelt dem Projekt an Geld. Aus Projektsicht notwendige Investitionen in Hardware, Software oder Wissen (in Form von Schulungen oder weiteren Mitarbeitern) unterbleiben.
- *Zu wenig Wissen und Erfahrung:* Es mangelt dem Projektteam an Wissen und Erfahrung in einigen Bereichen der Entwicklung. Beispielsweise wird ein neues, dem Team unbekanntes Entwicklungswerkzeug eingesetzt.

In seinem Buch „Death March“ ([Yourdon99]) gibt Edward Yourdon einige Ratschläge, wie Sie in schwierigen Projekten Ihr eigenes fachliches Überleben sichern. Falls die Situation Ihrer Ansicht nach völlig aussichtslos erscheint, sollten Sie dort nach Rettungsvorschlägen suchen.

Für die in der Praxis häufig vorkommenden „normal kritischen“ Situationen gebe ich Ihnen bezüglich der Softwarearchitektur folgende Ratschläge:

- Versuchen Sie, Risiken durch iteratives Vorgehen möglichst frühzeitig zu erkennen und zu adressieren. Iteration und Feedback verschafft Ihnen (hoffentlich) mehr Zeit, die garantiert auftretenden Probleme zu erkennen und zu lösen.
- Arbeiten Sie bei organisatorischen Problemen eng mit der Projektleitung oder dem Projektmanagement zusammen.
- Sie als Softwarearchitekt können wertvolle Hinweise zum Risikomanagement des Projekts liefern. Auf diese Weise können manche der Einflussfaktoren verhandelt werden (was Ihnen wiederum mehr Spielraum beim Entwurf der Architektur lässt).
- Sie können gemeinsam mit der Projektleitung alternative Modelle der Auslieferung oder Fertigstellung erarbeiten. Beispiel: Am vereinbarten Endtermin wird lediglich ein Teil der gewünschten Funktionalität geliefert.
- Sie können gemeinsam mit der Projektleitung die Auswirkung kritischer Anforderungen mit dem Auftraggeber diskutieren (und hier ebenfalls eine Entspannung der Situation erreichen).
- Falls sich die organisatorischen Probleme direkt auf technische Aspekte des Systems oder einzelne Bestandteile auswirken: Nutzen Sie diese Sachverhalte für Verhandlungen mit Projektleitung, Auftraggebern und Kunden.
- Prüfen Sie mit dem Auftraggeber, ob eine Revision der *Make-or-buy*-Entscheidung die Situation entschärfen kann. Gegebenenfalls kann auch der Zukauf einzelner Komponenten oder Systembestandteile helfen.
- Überarbeiten Sie mit dem Auftraggeber die Anforderungen an das System. Verhandeln Sie über andere Prioritäten der Anforderungen, um die Einschränkungen durch einige Einflussfaktoren zu lockern. Besonders kritisch und aufwendig sind (oftmals überzogene) Anforderungen an Verfügbarkeit und Performance.

Kooperation mit Projektleitung und Risikomanagement

Bedenken Sie bei organisatorischen Problemen einige wichtige Erfahrungen von Softwareprojekten:

- Niemand arbeitet unter hohem Druck besonders produktiv. Und keiner denkt unter Druck schneller. Im Gegenteil: Hoher Druck erhöht die Fehlerquote und die Bereitschaft zu „schmutzigen Tricks“.
- Einem verspäteten Projekt mehr Mitarbeiter zu geben, verzögert das Projekt zusätzlich ([Brooks95]).
- Weitere Probleme tauchen von allein auf. Bauen Sie daher in alle Schätzungen Sicherheitszuschläge ein. Unterschätzen Sie sich lieber.
- Wenn die Politik nicht mitspielt, wird das System niemals laufen.

Weiter mit Entwurf, Sichten und Dokumentation!

Im folgenden Kapitel 4 lernen Sie Grundlagen und Methoden des Entwurfs kennen – Sie finden die wesentlichen Entwurfsprinzipien und viele nützliche Heuristiken („Faustregeln“) für Ihre Arbeit.

Im Kapitel 5 stelle ich Ihnen die wichtigsten *Sichten* auf Softwarearchitekturen vor. Die sollten Sie kennen, wenn Sie Ihre eigenen Entwürfe kommunizieren und dokumentieren wollen.

3.7 Weiterführende Literatur

Die hier vorgestellten Ansätze, Einflussfaktoren zu gruppieren, zu priorisieren und mit spezifischen Strategien zu adressieren, werden in einigen anderen Disziplinen seit langem erfolgreich angewandt, etwa im Risikomanagement von Projekten oder dem *System Engineering* (siehe [Rechtin2000]). [Bass+03] diskutieren Qualitätsmerkmale und Möglichkeiten zur Umsetzung als sogenannte *Architectural Tactics*.

[Hofmeister2000] beschreibt eine „globale Analyse“, die auch auf die Darstellung der Einflussfaktoren sowie die Entwicklung von Strategien abzielt.

Das pragmatische und überaus lesenswerte Buch von Stefan Toth über Vorgehensmuster [Toth-13] gibt konkrete Anregungen, wie Sie Ihre „Pferdestärken auf die Straße“ bringen können – insbesondere im agilen Umfeld. Unbedingte Leseempfehlung!

[Martin08] zeigt den durchweg positiven Effekt von *Clean Code* auf: Verständlicher Quellcode hilft auch beim Verständnis von Architekturen.

[Rechtin2000] ist eine Fundgrube für Heuristiken und Vorgehensweisen beim Entwurf von Architekturen.