Basiswissen für Softwarearchitekten

Aus- und Weiterbildung nach iSAQB-Standard zum Certified Professional for Software Architecture – Foundation Level

1 Einleitung

Software ist allgegenwärtig. Dies gilt sowohl für kommerzielle Unternehmenssoftware als auch für nahezu alle anderen Bereiche des beruflichen, öffentlichen und privaten Alltags: Fliegen, Telefonieren, Überweisen, Autofahren – all das wäre ohne Software kaum noch möglich. In jedem Haushalt und in vielen Alltagsgegenständen, von der Waschmaschine bis zum Auto, werden softwaregesteuerte Bestandteile verwendet [BJ++06]. Software steht in der Regel nicht autark für sich, sondern ist in Geräte mit Hardware und Elektronik oder in Geschäftsprozesse, mit denen Unternehmen ihre Wertschöpfung erzielen, eingebettet [TTL00].

Der Nutzen und wirtschaftliche Erfolg von Unternehmen und Produkten wird zunehmend von Software und deren Qualität bestimmt (siehe [BM++96], [SV99], [TTL00]). Als Folge stehen Softwareingenieure und damit die Disziplin Software Engineering vor der Herausforderung, immer komplexere Anforderungen immer schneller und kostengünstiger bei gleichzeitig hoher Softwarequalität umzusetzen.

Die kontinuierliche Steigerung der Größe und Komplexität von softwareintensiven Systemen hat inzwischen dazu geführt, dass sie zu den komplexesten von Menschen geschaffenen künstlichen Systemen überhaupt zählen. Bestes Beispiel ist das Internet: ein auf Software basierendes weltumspannendes System. Inzwischen ist das Internet sogar auf der internationalen Raumstation ISS verfügbar und hat damit die Grenzen der Erde überschritten.

Nur ein strukturiertes und systematisches Herangehen kann dabei gesichert zum Erfolg führen. Trotz Anwendung etablierter Softwareentwicklungsmethoden bleibt die Anzahl der fehlgeschlagenen Softwareprojekte seit Jahren erschreckend hoch. Um dem entgegenzuwirken, versucht man in den frühen Phasen des Software Engineering bereits möglichst viele Fehler zu vermeiden bzw. dort zu identifizieren und auszumerzen. Zu diesen Phasen zählen insbesondere das Requirements Engineering sowie die Softwarearchitektur. Getreu den Worten von Ernst Denert, einem der Väter der methodischen Softwareentwicklung, wollen wir uns hier mit Softwarearchitektur beschäftigen, der »Königsdisziplin des Software Engineering« (zitiert aus dem Geleitwort von Ernst Denert in [Sie04]).

1.1 Softwarearchitektur als Disziplin im Software Engineering

Bereits in den 60er-Jahren wurden die Probleme mit Softwareprojekten unter dem Stichwort Softwarekrise bekannt. 1968 fand in Garmisch eine NATO-Konferenz hochrangiger Forscher und Praktiker statt, um unter dem Titel »Software Engineering« über die Zukunft der Softwareentwicklung nachzudenken. Heute gilt diese Konferenz als Geburtsstunde des Software Engineering [Dij72].

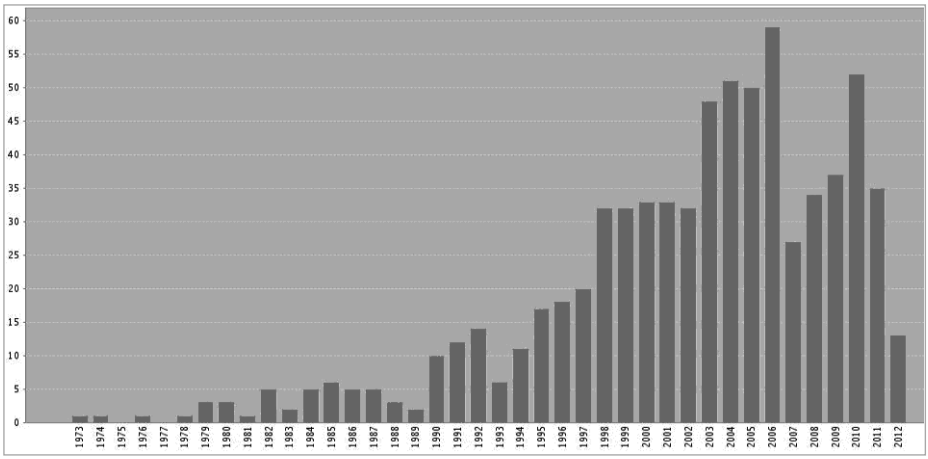

Abb. 1–1 *Veröffentlichungen zu Softwarearchitektur seit 1973 [Reu12]*

Im Vergleich zu traditionellen Ingenieurdisziplinen wie beispielsweise dem Bauwesen, das auf mehrere Tausend Jahre Erfahrung zurückblicken kann, ist Software Engineering mit dem Geburtsjahr 1968 noch sehr jung. So erscheint es auch nicht verwunderlich, dass dessen Teildisziplin Softwarearchitektur noch deutlich jünger ist. Abbildung 1–1 demonstriert dies deutlich: Das Web of Knowledge, eine der großen und renommierten Publikationsdatenbanken, verzeichnet erst ab den 90er-Jahren eine wachsende Anzahl von Publikationen zum Thema Softwarearchitektur [Reu12].

Betrachten wir hingegen die klassische Architektur im Bauwesen, so können wir auf eine bereits Jahrtausende währende Tradition zurückblicken. Ein wichtiger Vordenker war hier Marcus Vitruvius Pollio, ein römischer Architekt aus dem ersten Jahrhundert vor Christus. Er ist Autor des Werkes »De architectura«, das heute unter dem Titel »Ten Books on Architecture« bekannt ist [Vit60]. Vitruvius vertrat die These, dass gute Architektur durch eine kunstvolle Kombination der folgenden Elemente zu erreichen sei:

- **utilitas (Nützlichkeit):**
  Das Gebäude erfüllt seine Funktion.
- **firmitas (Festigkeit):**
  Das Gebäude ist stabil und langlebig.
- **venustas (Schönheit):**
  Das Gebäude ist ästhetisch gestaltet.

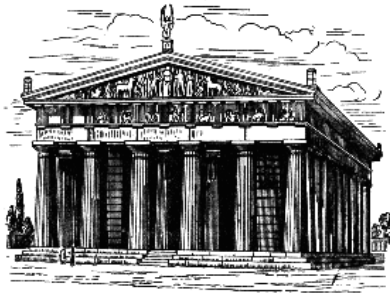

Abb. 1–2 *Architektur im alten Rom*

Diese These lässt sich direkt auf die Disziplin Softwarearchitektur übertragen. Ziel der Softwarearchitektur und damit Aufgabe eines Softwarearchitekten ist es, ein System zu konstruieren, das in einem kunstvoll ausgewogenen Dreiklang die drei folgenden Eigenschaften vereint:

- **utilitas (Nützlichkeit):**
  Die Software erfüllt die funktionalen und nicht funktionalen Anforderungen der Nutzer und Kunden.
- **firmitas (Festigkeit):**
  Die Software ist stabil im Hinblick auf die geforderten Qualitätseigenschaften, z.B. die Anzahl der gleichzeitig zu bedienenden Nutzer, und langlebig, da zukünftige Weiterentwicklungen möglich sind, ohne das System komplett neu bauen zu müssen.
- **venustas (Schönheit):**
  Die Software ist sowohl außen (gegenüber dem Nutzer) wohlstrukturiert, sodass sie intuitiv nutzbar ist, als auch innen (gegenüber demjenigen, der die Software pflegen und weiterentwickeln soll) wohlstrukturiert, sodass dieser die internen Strukturen der Software leicht verstehen und damit gut seinen Aufgaben nachkommen kann.

1.2 iSAQB – International Software Architecture Qualification Board

Softwarearchitektur ist eine junge Disziplin, über deren Umfang und Ausgestaltung in der Informatik trotz vieler Publikationen immer noch unterschiedliche Meinungen kursieren. Aufgaben und Verantwortungsbereiche von Softwarearchitekten werden unterschiedlich definiert und in Softwareprojekten ständig neu verhandelt.

Für andere Disziplinen im Software Engineering hingegen, wie z.B. beim Projektmanagement, Requirements Engineering oder Testen, gibt es inzwischen einen deutlich ausgereifteren Wissenskanon. Dafür bieten unabhängige Organisationen Lehrpläne an, die klar beschreiben, welche Kenntnisse und Fähigkeiten eine entsprechende Ausbildung vermitteln soll (Testen: *www.istqb.org*, Requirements Engineering: *www.ireb.de*, Projektmanagement: *www.pmi.org*).

Vor diesem Hintergrund haben Anfang 2008 verschiedene Softwarearchitekturexperten aus Wirtschaft und Wissenschaft das »International Software Architecture Qualification Board« als eingetragenen Verein (iSAQB e.V., *www.isaqb.org*) gegründet. Dessen Ziel ist es, Standards für die Ausbildung und Zertifizierung von Softwarearchitekten zu definieren. Bewusst wird im iSAQB jegliche Hersteller- oder Produktorientierung vermieden. Zertifizierungen auf den unterschiedlichen Stufen Foundation Level, Advanced Level und Expert Level ermöglichen es Softwarearchitekten, sich den Stand ihrer Kenntnisse und Fähigkeiten durch ein anerkanntes Verfahren bescheinigen zu lassen (siehe Abb. 1–3).

Abb. 1–3 *iSAQB-Zertifizierungsstufen (www.isaqb.org)*

Von diesem standardisierten Lehr- und Ausbildungsplan profitieren sowohl etablierte als auch angehende Softwarearchitekten und ebenso Unternehmen oder auch entsprechende Aus- und Weiterbildungseinrichtungen, da er die eingangs geschilderte begriffliche Unsicherheit beseitigt. Nur auf Basis von präzisen Lehr- und Ausbildungsplänen kann eine Prüfung und Zertifizierung angehender Softwarearchitekten stattfinden und so letztlich ein qualitätsgesicherter Ausbildungsstand von Softwarearchitekten mit einem entsprechend akzeptierten Wissenskanon etabliert werden.

Die Zertifizierung zum **Certified Professional for Software Architecture** (CPSA) wird von unabhängigen Zertifizierungsstellen durchgeführt. Basis für die Zertifizierung zum CPSA (Foundation Level) ist ein anspruchsvoller, vom iSAQB in Einklang mit dem Lehrplan entwickelter, nicht öffentlicher Fragenkatalog, aus dem eine Teilmenge als Prüfungsfragen ausgewählt wird. Für die Zertifizierung zum Advanced Level werden neben der Erfordernis des Besuches von lizenzierten Schulungen bzw. der Anerkennung eines anderen, nicht durch den iSAQB definierten Zertifikats praktische Aufgaben gestellt. Der Expert Level befindet sich derzeit noch in Entwicklung.

Auf Basis dieses Lehrplans bieten verschiedene lizenzierte Schulungsveranstalter mehrtägige Kurse an, die Wissen in diesen Themengebieten auffrischen und vielfach deutlich vertiefen. Die Teilnahme an einem Kurs wird zwar nachdrücklich empfohlen, ist jedoch nicht Bedingung für die Prüfungsanmeldung zur Zertifizierung.

1.3 Certified Professional for Software Architecture – Foundation und Advanced Level

Der iSAQB hat inzwischen nicht nur die Zertifizierungsrichtlinien für den CPSA Foundation Level, sondern auch für den Advanced Level definiert.

Der Advanced Level ist modular aufgebaut und besteht aus einzelnen Schulungen, die sich jeweils einem bestimmten Schwerpunkt der Kompetenz eines IT-Professionals widmen:

- **Methodische Kompetenz:**
  Wissen und Fähigkeiten im Bereich des systematischen Vorgehens bei IT-Projekten, unabhängig von Technologien
- **Technische Kompetenz:**
  Wissen und Fähigkeiten im Bereich des Einsatzes von Technologien zur Lösung von Entwurfsaufgaben
- **Kommunikative Kompetenz:**
  Wissen und Fähigkeiten im Bereich der Kommunikation, Präsentation, Rhetorik und Moderation zur effektiven Wahrnehmung der Rolle im Softwareentwicklungsprozess

Voraussetzungen für den Advanced Level sind:

- Ausbildung und Zertifizierung zum CPSA-F (Foundation Level)
- Mindestens 3 Jahre Berufserfahrung in der IT-Branche
- Mitarbeit an Entwurf und Entwicklung von mindestens zwei verschiedenen IT-Systemen
- Für die Prüfung: mindestens 70 Credit Points aus allen drei Kompetenzbereichen (jeweils mindestens 10 Credit Points)

Die Prüfung besteht aus der Bearbeitung einer Prüfungsaufgabe in Eigenregie und der anschließenden Besprechung der Lösung mit zwei unabhängigen Prüfern in einem Interview.

Für den Foundation Level wurden die Bereiche, in denen ein Softwarearchitekt über fundiertes Wissen und Fähigkeiten verfügen sollte, im Rahmen eines öffentlich zugänglichen Lehrplans beschrieben [isaqb-lehrplan]. Danach soll angehenden Softwarearchitekten folgendes Spektrum an Inhalten vermittelt werden:

- der Begriff und die Bedeutung von Softwarearchitektur,
- die Aufgaben und Verantwortungsbereiche von Softwarearchitekten,
- die Rolle des Softwarearchitekten in Projekten,
- State-of-the-Art-Methoden und -Techniken zur Entwicklung von Softwarearchitekturen.

Im Mittelpunkt steht der Erwerb folgender Fähigkeiten:

- mit anderen Projektbeteiligten aus den Bereichen Anforderungsmanagement, Projektmanagement, Test und Entwicklung wesentliche Softwarearchitekturentscheidungen abzustimmen,
- Softwarearchitekturen auf Basis von Sichten, Architekturmustern und technischen Konzepten zu dokumentieren und kommunizieren,
- die wesentlichen Schritte beim Entwurf von Softwarearchitekturen zu verstehen und für kleine und mittlere Systeme selbstständig durchzuführen.

Die Schulung zum Foundation Level vermittelt das notwendige Wissen, um für kleine und mittlere Systeme ausgehend von einer hinreichend detailliert beschriebenen Anforderungsspezifikation eine dem Problem angemessene Softwarearchitektur zu entwerfen und zu dokumentieren. Diese kann dann als Implementierungsgrundlage bzw. -vorlage genutzt werden. Teilnehmer erhalten das Rüstzeug, um problembezogene Entwurfsentscheidungen auf der Basis ihrer vorab erworbenen Praxiserfahrung zu treffen.

Abbildung 1–4 zeigt die inhaltliche Struktur und die Gewichtung der einzelnen Bereiche des Lehrplans für den iSAQB Certified Professional for Software Architecture (CPSA), Foundation Level.

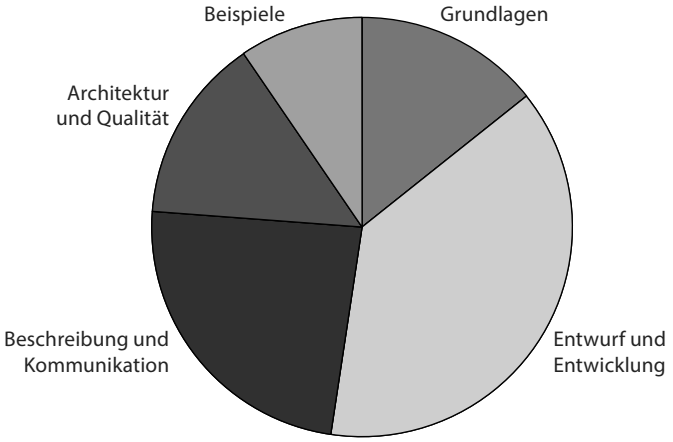

Abb. 1–4 *Struktur des iSAQB-Lehrplans für CPSA, Foundation Level*

Sie haben die Möglichkeit, sich bei verschiedenen unabhängigen Anbietern durch eine Prüfung gemäß dem iSAQB-Lehrplan zertifizieren zu lassen. Für die Zertifizierung setzen die Prüfungsanbieter standardisierte Prüfungsfragen ein, die der iSAQB erarbeitet hat.

Für die Prüfungen wird ein Multiple-Choice-Verfahren verwendet. Entsprechend objektiv ist das Prüfungsergebnis messbar.

Mit der Prüfung können Sie somit Ihr notwendiges Grundlagenwissen als Softwarearchitekt nachweisen. Natürlich müssen Sie dann später in der Anwendung zeigen, dass Sie Ihr Wissen auch praktisch und erfolgreich in konkreten Architekturen einzusetzen wissen.

1.4 Zielsetzung des Buches

Wir, das Autorenteam des Buches, haben gemeinsam mit anderen iSAQB-Mitgliedern am iSAQB-Lehrplan für den Certified Professional for Software Architecture, Foundation Level, gearbeitet. Im Rahmen dieser Zusammenarbeit ist auch die Idee zu diesem Buch entstanden. Dementsprechend verfolgen wir darin die zentrale Zielsetzung, kompakt und prägnant das notwendige Wissen für die CPSA-Prüfung, Foundation Level, und somit das Fundament für den Wissenskanon in der Disziplin Softwarearchitektur bereitzustellen. Das Buch ist demzufolge die ideale Referenz für eine entsprechende Prüfungsvorbereitung. Wir empfehlen Ihnen ergänzend den Besuch entsprechender Schulungen, da dort das Lehrmaterial durch über dieses Buch hinausgehende praktische Beispiele von Softwarearchitekturen und persönliche Erfahrungen der jeweils Lehrenden abgerundet wird.

Da der iSAQB und somit auch das Buch primär auf methodische Fähigkeiten und Wissen fokussiert, gehören konkrete Implementierungstechnologien oder spezielle Werkzeuge explizit nicht zum standardisierten Lehrinhalt. Deshalb haben wir

dieses Buch bewusst technologieneutral verfasst. Auch die von uns verwendeten Notationen, wie z. B. die UML, sind nur exemplarisch zu verstehen. Ebenso ist es nicht Ziel des Buches, ein einzelnes konkretes Vorgehensmodell oder einen spezifischen Entwicklungsprozess darzustellen. Vielmehr werden von uns an vielen Stellen mehrere Beispiele etwa für Notationen oder Vorgehensmodelle kurz vorgestellt.

In diesem Buch erklären wir vor allem wichtige Begriffe und Konzepte der Softwarearchitektur und stellen deren Bezug zu anderen Disziplinen dar. Darauf aufbauend führen wir die grundlegenden Techniken und Methoden für den Entwurf und die Entwicklung, die Beschreibung und Kommunikation sowie die Qualitätssicherung von Softwarearchitekturen ein. Schließlich betrachten wir die Rolle, die Aufgaben, das Umfeld und die Arbeitsumgebung von Softwarearchitekten und deren Einbettung in die umfassende Organisations- und Projektstruktur.

1.5 Voraussetzungen

Entsprechend der oben genannten Zielsetzung setzt das vorliegende Buch – wie auch der iSAQB-Lehrplan – Erfahrung in der Softwareentwicklung voraus. Insbesondere gehören folgende Inhalte nicht zum Lehrplan und sind damit auch nicht Thema des Buches, obgleich sie zu den notwendigen Kompetenzen von Softwarearchitekten zählen:

- typischerweise mehrjährige praktische Erfahrung in der Softwareentwicklung, erworben durch Programmierung unterschiedlicher Systeme,
- vertiefte Kenntnisse und praktische Erfahrung mit mindestens einer höheren Programmiersprache,
- Grundlagen der Modellierung und Abstraktion sowie der Modellierungssprache UML, insbesondere der Klassen-, Paket-, Komponenten- und Sequenzdiagramme sowie deren Bezug zum Quellcode,
- praktische Erfahrung in technischer Dokumentation, insbesondere in der Dokumentation von Quellcode, Systementwürfen oder technischen Konzepten,
- Kenntnisse über die Methodik beim Testen von Software in den verschiedenen Teststufen.

Darüber hinaus sind Kenntnisse und Erfahrung mit der Objektorientierung für das Verständnis einiger Konzepte hilfreich. Ebenfalls wünschenswert ist Erfahrung in der Konzeption und Implementierung verteilt ablaufender Anwendungen wie etwa Client-Server-Systeme oder Webanwendungen.

1.6 Leitfaden für den Leser

Der Aufbau dieses Buches orientiert sich vor allem an der Struktur und den Inhalten des iSAQB-Lehrplans Foundation Level nach Abbildung 1–4 bzw. [isaqb-lehrplan]:

- In Kapitel 2 beschreiben wir grundlegende Begriffe und Inhalte des Themenbereichs Softwarearchitektur, die in den folgenden Kapiteln aufgegriffen und vertieft werden. Beispielsweise wird dort der Begriff der »Sicht« auf ein Softwaresystem im Rahmen einer Softwarearchitektur eingeführt.
- Aspekte des praktischen Entwurfs von Softwarearchitekturen behandeln wir in Kapitel 3. Themen sind dort Varianten des Vorgehens bei der Architekturentwicklung, wichtige Architekturmuster wie Schichten, Pipes and Filters, Model View Controller, Entwurfsprinzipien wie Kopplung, Kohäsion, Trennung von Verantwortlichkeiten u.a.m.
- Inhalt des Kapitel 4 sind ausgewählte praxisnahe Beschreibungsmittel sowie in der Praxis bewährte Richtlinien, die es Ihnen erlauben, Ihre Softwarearchitektur zu dokumentieren und zielgruppenorientiert anderen zu vermitteln. Das iSAQB-Sichtenmodell, Querschnittsaspekte in Softwarearchitekturen sowie praktisch bewährte Richtlinien für die Dokumentation von Softwarearchitekturen sind Beispiele für den Inhalt des Kapitels.
- In Kapitel 5 werfen wir einen ersten Blick auf den Zusammenhang von Softwarearchitektur und Qualitätsfragestellungen. Wichtige Begriffe dieses Abschnitts sind u.a. bezogen auf Software: Qualität, Qualitätsmerkmale, ATAM (Architecture Tradeoff Analysis Method), Qualitätsbaum, Kompromisse (bei der Umsetzung von Qualitätsmerkmalen), qualitative Architekturbewertung und Risiken bezüglich der Erreichung von Qualitätsmerkmalen.
- Abschließend zeigen wir in Kapitel 6 (Exkurs) eine Reihe von Beispielen für Unterstützungswerkzeuge des Softwarearchitekten, wie z.B. solche zur Modellierung, Generierung oder Dokumentation.
- Einige beispielhafte Übungsfragen, ein Glossar sowie ein Quellenverzeichnis runden das Buch ab.

Speziell zur iSAQB-Prüfungsvorbereitung sollten Sie vor allem die Kapitel 2 – 5 gründlich durcharbeiten und ergänzende Blicke in die anderen Abschnitte werfen. Ansonsten empfiehlt es sich, zumindest das Kapitel 2 komplett zu lesen und dann die Sie besonders interessierenden Themen zu vertiefen.

1.7 Zielpublikum

Als Zielpublikum dieses Buches sehen wir in erster Linie Zertifizierungsinteressierte, die es – ggf. neben Schulungen – als Vorbereitung zur Prüfung einsetzen wollen. Hinzu kommen Praktiker und Studierende, die die praktischen Grundbegriffe von Softwarearchitekturen kennenlernen wollen.

Interessant ist dieses Buch auch für Softwareprojektmanager, Softwareproduktmanager sowie Entscheider auf der mittleren Softwareentwicklungsebene als Einstiegsüberblick zum Thema Softwarearchitektur.

1.8 Danksagungen

Wir möchten uns an dieser Stelle beim iSAQB-Verein für seine unterstützende Mitwirkung bedanken. Frau Ingrid Schindler vom Lehrstuhl für Software Systems Engineering der Technischen Universität Clausthal und Mitarbeiter der ITech Progress haben uns beim Anfertigen der Abbildungen sehr unterstützt. Die Erstellung der 5. Auflage wäre ohne die unermüdliche Unterstützung von Noah Neukam und Thorsten Mayer von ITech Progress nicht möglich gewesen.

Unserer Betreuerin seitens des dpunkt.verlags, Frau Christa Preisendanz, danken wir für ihre Geduld.

Zu guter Letzt möchten wir ganz besonders unseren Familien und Partnern danken, die an zahllosen Tagen die Zeit und Geduld aufgebracht haben, uns gemeinsam an diesem Buch arbeiten zu lassen.

4.6 Übliche Dokumenttypen für Softwarearchitekturen

Zur Beschreibung der in den vorangegangenen Abschnitten dargestellten Architekturinformationen kommt üblicherweise eine Reihe verschiedener Dokumente zum Einsatz. Dieser Abschnitt gibt hierüber einen ersten Überblick.

4.6.1 Zentrale Architekturbeschreibung

Die zentrale Architekturbeschreibung ist (sinnvollerweise) das Kerndokument für eine Softwarearchitektur. Sie enthält soweit möglich alle architekturrelevanten Informationen, dazu gehören:

- Aufgabenstellung, Ziele (Vision), Qualitätsanforderungen und Stakeholderinnen
- Technische und organisatorische Rahmenbedingungen
- Sichten, Entscheidungen, verwendete Muster
- Technische Konzepte
- Qualitätsbewertungen
- Erkannte Risiken

Hier bietet sich der Einsatz eines passenden Schablonendokuments, also einer Art »Architektur-Dokumentations-Schablone« an. Zum Beispiel folgt die Dokumentenschablone unter [arc42] zu einem guten Teil den in den vorangegangenen Abschnitten dieses Kapitels vorgestellten Inhalten.

Eine zentrale Architekturbeschreibung kann durchaus einen größeren Umfang annehmen, sodass ihre Beschreibung (und Pflege) in einem einzelnen Dokument ggf. nur noch bedingt sinnvoll ist. Die Inhalte einer solchen Beschreibung können mit verschiedenen Werkzeugen verwaltet werden. Einige typische Optionen sind:

- **Dokumente:**
  Dokumente, die mit gängigen Textverarbeitungen erstellt werden können, sind in der Regel recht einfach zu nutzen und zu verwalten, solange sie nicht zu groß werden.
- **CASE-/MDA-/UML-Tools:**
  Modellierungswerkzeuge können mittels oft mächtiger Reportgeneratoren recht hilfreich bei der Erstellung von Dokumentationen sein (siehe auch Kap. 6). Nicht unterschätzt werden sollte allerdings (gerade bei kleineren Projekten) der Aufwand, den die projektspezifische Anfangskonfiguration solcher Werkzeuge annehmen kann. Von Vorteil ist der in der Wartung dann oft deutlich höhere Automatisierungsgrad für das wiederholte Generieren einer Architekturdokumentation, da z.B. UML-Diagramme oder gar Codestücke ohne »Copy & Paste« in eine neue Dokumentenversion eingebaut werden können.

- **HTML-Seiten oder Wikis:**
  Eventuell erlauben »potenziell etwas pragmatischere« Werkzeuge wie Wikis einen Mittelweg zwischen Dokumentation und Modellierungswerkzeugen.
- Beliebige Mischformen

In Kapitel 6 wird eine Reihe weiterer Softwarewerkzeuge behandelt, die für Softwarearchitekturen nützlich sein können.

4.6.2 Architekturüberblick

Der Architekturüberblick dient als schnell lesbare Kurzfassung (möglichst nicht mehr als 30 Seiten) der zentralen Architekturbeschreibung. Er betrachtet vergleichbare Inhalte, beschränkt sich aber auf die wesentlichen Punkte wie zentrale Sichten, Hauptqualitätsanforderungen und Kernentscheidungen.

Falls für die Erstellung einer ausführlichen zentralen Architekturbeschreibung, z.B. aus zeitlichen oder Aufwandsgründen, keine Möglichkeit besteht, kann der Architekturüberblick als Minimalersatz für eine komplette Beschreibung dienen.

4.6.3 Dokumentübersicht

Die Dokumentübersicht ist ein Verzeichnis, das (pro Projekt bzw. pro zu beschreibender Softwareanwendung) als Index für alle jeweils architekturrelevanten Dokumente dient und auch deren Abhängigkeiten nennt. Sinnvollerweise sollten organisatorische Richtlinien festgelegt werden, wie dieses Verzeichnis auszusehen hat und an welcher Stelle es zu finden ist. Zudem sollten Hinweise enthalten sein, welche Dokumente von wem, also von welcher Projektrolle, in welcher Reihenfolge zu lesen sind.

4.6.4 Übersichtspräsentation

Eine Übersichtspräsentation ist ein Foliensatz, anhand dessen die Architektur in maximal einer Stunde (technisch) präsentiert werden kann.

Eine managementtaugliche Variante davon sollte die zentralen Aussagen und insbesondere auch den Geschäftsnutzen in 10 Minuten zusammenfassen können.

4.6.5 »Architekturtapete«

Mit einer »Architekturtapete« sollen viele Architekturaspekte in einem Gesamtüberblick dargestellt werden. Praktisch ist dies oft eine Sammlung von Postern z.B. mit Sichtendarstellungen zuzüglich Verfeinerungen, Qualitätsaspekten usw. Diese werden gemeinsam an einer Wand oder auf Metaplanwänden aufgehängt und erlauben die interaktive Diskussion z.B. zwischen Architektur und Entwicklung über spezielle Themen.

Achtung: Eine »Architekturtapete« kann, als Diskussionsgrundlage eingesetzt, sehr hilfreich sein. Zu vermeiden ist hingegen, sie dogmatisch zu sehen, denn dann wirkt sie schnell abschreckend auf diejenigen, die das Softwaresystem umsetzen, testen und betreiben sollen.

4.6.6 Handbuch zur Dokumentation

Das Handbuch erläutert die Funktionsweise und die Struktur der gesamten Dokumentation. Es ist auch der geeignete Ort, um Notationen gesammelt zu erläutern.

4.6.7 Architecture Decision Record

Ein Architecture Decision Record oder auch ein »ADR« ist ein festes Format zur Dokumentation von getroffenen Architekturentscheidungen. In einem ADR wird eine Entscheidung in einem eigenen Dokument oder Abschnitt isoliert betrachtet, mit dem Ziel, eine zugänglichere Dokumentation zu erhalten. Der Fokus liegt dabei nicht nur auf dem Ergebnis der Entscheidung, sondern insbesondere auf der Dokumentation der getroffenen Annahmen, Beweggründe und Einflussfaktoren zum Zeitpunkt der Entscheidung [Ny11]. Ein ADR, wenn korrekt eingesetzt, hilft dabei, eine Dokumentation zielgruppengerecht, korrekt und nachvollziehbar zu gestalten. Die Erfindung des Formats wird Michael Nygard zugeschrieben, der die Struktur 2011 definierte. Ein ADR besteht aus den folgenden Abschnitten (siehe [Ny11]):

- **Titel:**
  Er soll eine aussagekräftige Auskunft über das Thema der Entscheidung geben.
- **Kontext:**
  Er soll die Einflüsse auf die Entscheidung beschreiben. Der Abschnitt soll eine akkurate Abbildung der Umstände sein und Randbedingungen oder vorherige Entscheidungen auflisten, die nun zur aktuellen Entscheidungsnotwendigkeit führen.
- **Entscheidung:**
  Dieser Abschnitt liefert die »Antwort« auf den vorherigen Abschnitt. Hier werden oft auch die zur Verfügung stehenden Diskussionen aufgelistet und diskutiert.
- **Status:**
  Er gibt Auskunft darüber, ob eine Entscheidung in Kraft ist, lediglich einen Vorschlag darstellt oder vielleicht schon überholt ist.
- **Konsequenzen:**
  Dieser Abschnitt soll beschreiben, wie sich der Kontext durch die Entscheidung verändert. Auch hier soll eine möglichst ehrliche Beschreibung inklusive positiver und negativer Auswirkungen der Entscheidung stattfinden, um Nachvollziehbarkeit zu gewährleisten.

4.6.8 Technische Informationen

Hierbei handelt es sich um ein oder mehrere Dokumente mit wichtigen Informationen für Projektentwickler und Tester. Darin sollten Informationen zu den Entwicklungsmethoden, Programmierrichtlinien sowie zum Bau, Start und Test des Systems hinterlegt sein (vgl. hierzu auch Abschnitt 4.5).

4.6.9 Dokumentation von externen Schnittstellen

Ein besonderes Augenmerk sollte auf die Dokumentation von insbesondere extern sichtbaren Schnittstellen des Softwaresystems gelegt werden. Diese sind für das Zusammenwirken des Gesamtsystems in seinem Kontext von zentraler Bedeutung.

Leider werden externe Schnittstellen in realen Projekten (allzu) häufig zu »Zeit fressenden Problemstellen«. Widmen Sie ihnen eher früher als später die nötige Aufmerksamkeit, denn gerade dort »liegt der Teufel oft im Detail«. Folglich macht sich ein etwas stärkerer und recht frühzeitiger Beschreibungsaufwand an dieser Stelle oft bezahlt (deutlich mehr als »ganz besonders schöne Diagramme« oder »das letzte Quäntchen syntaktischer UML-Feinheit« in einer Sichtenbeschreibung).

4.6.10 Template

Bei Schnittstellenbeschreibungen bietet es sich an, wieder einem gleichbleibenden Muster zu folgen. Tabelle 4–9 stellt eine Reihe typischer Elemente aus Schnittstellenbeschreibungen zusammen:

Überschrift	Inhalt
Identifikation	Genaue Bezeichnung und Version der Schnittstelle
Bereitgestellte Ressourcen	Welche Ressourcen stellt dieses Element bereit? ■ Syntax der Ressource: API, Methodensignaturen (z.B. OMG-IDL, WSDL) ■ Semantik der Ressource: Welche Auswirkungen hat ein Aufruf dieser Ressource? • ausgelöste Events • geänderte Daten • geänderte Zustände • sonstige wahrnehmbare Nebenwirkungen • Restriktionen bei der Benutzung der Ressource
Fehlerszenarien	Beschreibung der Fehlersituation und deren Behandlung
Variabilität und Konfigurierbarkeit	Kann das Verhalten verändert oder konfiguriert werden (beispielsweise über Konfigurationsparameter)?
Qualitätseigenschaften	Welche Qualitätseigenschaften (Verfügbarkeit, Performance, Sicherheit, Parallelisierbarkeit etc.) gelten für diese Schnittstelle?
Entwurfs-entscheidungen	Welche Gründe haben zum Entwurf dieser Schnittstelle geführt? Welche Alternativen gibt es und warum wurden sie verworfen?
Hinweise	Hinweise oder Beispiele zur Benutzung

Tab. 4–9 *Typische Elemente aus Schnittstellenbeschreibungen*

4.7 Praxisregeln zur Dokumentation

Für jedwede Art von Architekturdokumentation gibt es – wie für viele andere vorgehens- oder technisch orientierte Dokumentationen – eine Reihe von bewährten Regeln. Diese dienen vor allem der praxistauglichen Lesbarkeit und Zweckdienlichkeit solcher Dokumentationen.

4.7.1 Regel 1: »Schreiben aus der Sicht der Leserin«

Versuchen Sie Ihre Dokumentation aus der Betrachtungsweise Ihrer Leserinnen zu erstellen, denn sonst fühlen sich diese in ihren Anforderungen nicht ernst genommen. Naturgemäß werden Dokumente häufiger gelesen als geschrieben – und Dokumentation wird tatsächlich gelesen.

Versuchen Sie zu starken Fachjargon zu vermeiden (in der Praxis ist dies in Dokumenten stets eine Gratwanderung). Besonders zentrale Fachbegriffe sollten separat erläutert werden, z. B. in Form eines Glossars.

Augenmerk sollten Sie auf die Struktur von Dokumenten legen, um Ihre Gedanken und Ideen in eine zielführende Reihenfolge zu bringen. Einmal mehr ist hier die Verwendung passender Dokumentenschablonen zu empfehlen.

4.7.2 Regel 2: »Unnötige Wiederholung vermeiden«

Versuchen Sie soweit möglich Wiederholungen zu vermeiden bzw. setzen Sie sie wenn nötig nur sparsam und gezielt ein. Im Ergebnis

- vereinfachen Sie dadurch normalerweise die Verwendung der Dokumentation und
- vereinfachen deren Pflege bzw. Änderungen merklich.

Stellen Sie sich bei Wiederholungen (auch in leichter Variation) die Fragen:

- Ist die Variation Absicht?
- Ist dieser Variation Bedeutung beizumessen?

Dokumentation aus unterschiedlichen Sichtweisen ist allerdings typischerweise nicht als Wiederholung einzuordnen, sondern dient vielmehr zur Vertiefung des Verständnisses über einen Sachverhalt.

4.7.3 Regel 3: »Mehrdeutigkeit vermeiden«

Architekturdokumentation lässt häufig bewusst spätere Designentscheidungen als geplanten Handlungsspielraum offen. Zu viel Interpretationsspielraum bei Architekturvorgaben führt jedoch auch leicht zu ungeplanter Mehrdeutigkeit.

Um hier Abhilfe zu schaffen, könnten formale Beschreibungssprachen eingesetzt werden. Allerdings ist deren Einsatz in der Praxis (zu) selten.

Werden symbolische Notationen verwendet (wie z. B. UML), so sollten Sie die Bedeutung der Symbole erklären oder eine Referenz auf die Begriffserklärung einbauen.

Mehrdeutigkeit entsteht auch wenn sich mündliche und schriftliche Dokumentation nicht decken. Achten Sie daher darauf, identische Begriffe und Argumentationen zu verwenden.

4.7.4 Regel 4: »Standardisierte Organisationsstruktur bzw. Schablonen«

Gerade wenn Sie häufiger Architekturdokumente schreiben, ist deren (einheitliche) Struktur von Wichtigkeit. Sie bietet Ihren Leserinnen einen Wiedererkennungswert. Zudem vereinfacht sie die Referenzierbarkeit von Dokumentteilen (z. B. in der Form: »Siehe Bausteinsicht aus dem Kunden-Management-System«).

Wenn die Struktur festgelegt ist, sollte die Leserin darüber informiert werden. Anhand einer festen Struktur ergibt sich leicht eine Übersicht über bereits abgeschlossene und noch zu erledigende Teile der Dokumentation. Ebenfalls unterstützt wird die Qualitätssicherung der Dokumentation, da alle von Dokumenten abzudeckenden Aspekte vorab definiert sind.

4.7.5 Regel 5: »Begründen Sie wesentliche Entscheidungen schriftlich«

Um Ihre Leserinnen dabei zu unterstützen, Ihre Architekturen und Entwürfe nachzuvollziehen, ist es sehr hilfreich, wesentliche Entscheidungen kurz zu begründen. Begründungen können z. B. durch Verweise auf entsprechende Unternehmensrichtlinien erfolgen (».NET oder Java EE ist bevorzugt für Anwendungen der Art X einzusetzen« oder: »Die Datenhaltung der Kundenstammdaten erfolgt ausschließlich auf dem Mainframe in der zentralen Kundendatenbank«).

Ebenfalls von Interesse können explizit verworfene andere Optionen sein, aber auch die Diskussion von Vor- und Nachteilen einer Lösung, z. B. in der Form: »Die Nutzung eines selbst entwickelten Persistenz-Frameworks mag zwar zu höherer Flexibilität führen, rechtfertigt aber gegenüber bestehenden Frameworks wie Y nicht den resultierenden Entwicklungs- und langfristigen Wartungsaufwand.«

Im Ergebnis kann u. a. Zeit bei Diskussionen gespart werden, wenn Entscheidungen unter neuen Bedingungen überdacht werden müssen. Insgesamt helfen Begründungen somit Ihren Leserinnen, Ihre Entscheidungen zu verstehen und für sich im Nachhinein nachzuvollziehen.

4.7.6 Regel 6: »Überprüfung auf Gebrauchstauglichkeit«

Ein bedeutsamer Punkt für Dokumentation ist der tatsächliche praktische Nutzen für Ihren Leserkreis. Bevor Sie eine Dokumentation abschließend veröffentlichen, sollten Sie daher unbedingt von passenden Vertreterinnen Ihrer Zielgruppe Reviews durchführen lassen – und diese anschließend einarbeiten. Letztlich kann nur der anvisierte Nutzerkreis entscheiden, ob die richtigen Informationen in der richtigen Weise dargeboten werden.

Neben den Reviews sollte auch der Reviewprozess an sich begutachtet werden. Installieren Sie also einen Verbesserungsprozess für Dokumentation, anhand dessen auch die Dokumentationsrichtlinien selbst regelmäßig auf Schwachstellen hin überprüft werden.

4.7.7 Regel 7: »Übersichtliche Diagramme«

Eine Regel – von der es allerdings durchaus des Öfteren begründbare Abweichungen geben kann – lautet: »Vermeiden Sie zu große Diagramme.« Ergebnissen der Kognitionswissenschaft zufolge sind fünf bis neun Elemente (7±2) in Diagrammen für Menschen noch gut überschaubar.

Beachten Sie: Die Architekturtapete aus Abschnitt 4.6.5 ist eine klare Ausnahme dieser Regel.

4.7.8 Regel 8: »Regelmäßige Aktualisierungen«

Aktualisieren Sie Ihre Dokumentation bzw. etablieren Sie einen Prozess, der die Dokumentation nicht nur während der Entwicklung regelmäßig anpasst, sondern dies auch als Teil von Wartungsarbeiten durchführt.

Mangelnde Aktualisierung …

- führt zu unvollständiger Dokumentation,
- lässt Nutzen und Nutzung der Dokumentation sinken,
- lässt Dokumentation sich nicht als maßgebliche Quelle für Informationen etablieren, und es erfolgt unnötiges Nachfragen.

Noch unbeantwortete Fragen in einer Dokumentation verlangen ebenfalls nach zeitnaher Aktualisierung. Ändern sich jedoch Designentscheidungen sehr schnell, so sollten Sie auch nicht zu oft aktualisieren (sonst tun Sie bald nichts anderes mehr …), sondern ggf. warten, »bis sich der Staub etwas gelegt hat«. Hilfreich ist es, hierfür relativ fixe Synchronisationszeitpunkte festzulegen.

4.7.9 EXKURS: Regel 9: »Passen Sie die Änderbarkeit der Dokumentation an die Architektur an«

Je nach Zeitpunkt im Lebenszyklus einer Architektur werden mehr oder weniger Änderungen an ihr vorgenommen. Zum Beispiel werden die ersten Entwürfe einer Architektur sich oft und schnell ändern, wohingegen Änderungen an einer seit Jahren bestehenden Architektur eher langsam und mit Bedacht durchgeführt werden [Kee17]. Sie können den Formalisierungsgrad der Dokumentation diesen Gegebenheiten anpassen und dadurch die Geschwindigkeit maximieren oder das Risiko einer unbedachten Änderung minimieren. Dafür stehen Ihnen unterschiedliche Stellschrauben zur Verfügung, wie zum Beispiel der Detaillierungsgrad der Dokumentation, das Format, die Zugänglichkeit, die Formalität der Diagramme oder die Länge und das Vorhandensein eines Prüfungs-, Änderungs- und Freigabeprozesses.

4.8 Beispiele weiterer Architektur-Frameworks

Neben dem in Kapitel 2 genannten umfassenden Standard ISO/IEC/IEEE 42010: 2011 und dem in den vorangegangenen Abschnitten beschriebenen pragmatischen iSAQB-Ansatz gibt es eine Vielzahl weiterer Ansätze zur Beschreibung von Softwarearchitekturen bzw. Architektur-Frameworks. Um Ihnen einen Eindruck davon zu vermitteln – und ggf. Neugier für weitere Recherche zu wecken –, zeigen wir in diesem Abschnitt in Kurzform einige Beispiele, auch hier ohne Anspruch auf Vollständigkeit.

Einige bekanntere Framework-Ansätze für Softwarearchitektur oder noch weiter gehende Ansätze für Unternehmensarchitektur (Enterprise Architecture) sind u.a.:

- 4+1 (Kruchten)
- Department of (US) Defense Architectural Framework (DoDAF)
- Model Driven Architecture der Object Management Group (OMG-MDA)
- Standards und Architekturen für E-Government-Anwendungen (SAGA) als Framework der deutschen Bundesverwaltung
- SAPs Enterprise Architecture Framework
- The Open Group Architecture Framework (TOGAF®)
- Zachman Framework (IBM)

In den folgenden Abschnitten gehen wir kurz auf zwei dieser Frameworks ein.