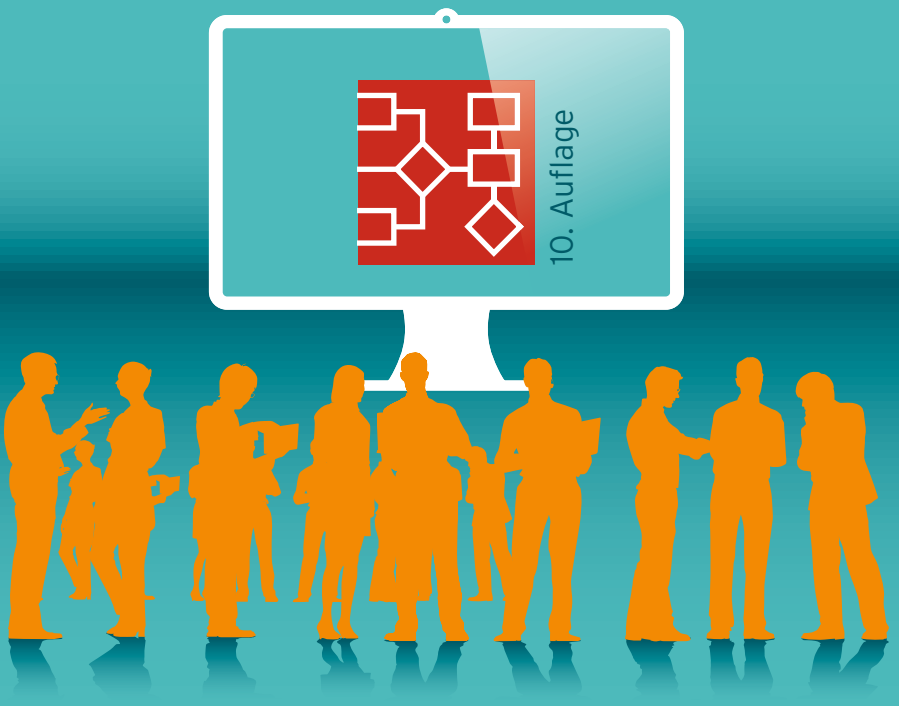


Effektive Softwarearchitekturen

Gernot STARKE

Ein praktischer Leitfaden

Ideal zur Vorbereitung
auf die iSAQB-Zertifizierung

HANSER

INNOQ

Starke

Effektive Softwarearchitekturen

Gernot Starke

Effektive Softwarearchitekturen

Ein praktischer Leitfaden

10., überarbeitete Auflage

HANSER

Der Autor:

Dr. Gernot Starke, Köln
INNOQ Fellow
www.gernotstarke.de

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *https://dnb.d-nb.de* abrufbar.

Lektorat: Brigitte Bauer-Schiewek, Kristin Rothe
Copy editing: Petra Kienle, Fürstenfeldbruck
Umschlagdesign: Marc Müller-Bremer, *www.rebranding.de*, München
Umschlagrealisation: Tom West, unter Verwendung von Grafiken für das Titelmotiv:
© shutterstock.com/Rawpixel.com und Vector VA
Layout: Manuela Treindl, Fürth
Druck und Bindung: CPI books GmbH, Leck
Printed in Germany

Print-ISBN: 978-3-446-47672-1
E-Book-ISBN: 978-3-446-47909-8
E-Pub-ISBN: 978-3-446-48277-7

Inhalt

1 Einleitung

Wir bauen Software wie Kathedralen:
Zuerst bauen wir – dann beten wir.

Gerhard Chroust

Bitte erlauben Sie mir, Sie mit einer etwas bösartigen kleinen Geschichte zur weiteren Lektüre dieses Buchs zu motivieren.

Eine erfolgreiche Unternehmerin möchte sich ein Domizil errichten lassen. Enge Freunde raten ihr, ein Architekturbüro mit dem Entwurf zu betrauen und die Erstellung begleiten zu lassen. Nur so ließen sich die legendären Probleme beim Hausbau (ungeeignete Entwürfe, mangelnde Koordination, schlechte Ausführung, Pfusch bei Details, Kostenexplosion und Terminüberschreitung) vermeiden.

Um die für ihr Vorhaben geeigneten Architekten zu finden, beschließt sie, einigen namhaften Büros kleinere Testaufträge für Einfamilienhäuser zu erteilen. Natürlich verrät sie keinem der Kandidaten, dass diese Aufträge eigentlich Tests für das endgültige Unterfangen sind.

Nach einer entsprechenden Ausschreibung in einigen überregionalen Tageszeitungen trifft unsere Bauherrin folgende Vorauswahl:

- Wasserfall-Architektur KG, Spezialisten für Gebäude und Unterfangen aller Art,
- V&V Architektur GmbH & Co. KG, Spezialisten für Regierungs-, Prunk- und Profanbauten,
- Extremarchitekten AG.

Alle Büros erhalten identische Vorgaben: Ihre Aufgabe besteht in Entwurf und Erstellung eines Einfamilienhauses (EFH). Weil unsere Unternehmerin jedoch sehr häufig, manchmal fast sprunghaft, ihre Wünsche und Anforderungen ändert, beschließt sie, die Flexibilität der Kandidaten auch in dieser Hinsicht zu testen.

Wasserfall-Architektur KG

Die Firma residiert im 35. Stock eines noblen Bürogebäudes. Dicke Teppiche und holzvertäfelte Wände zeugen vom veritablen Wohlstand der Firmeneigner.

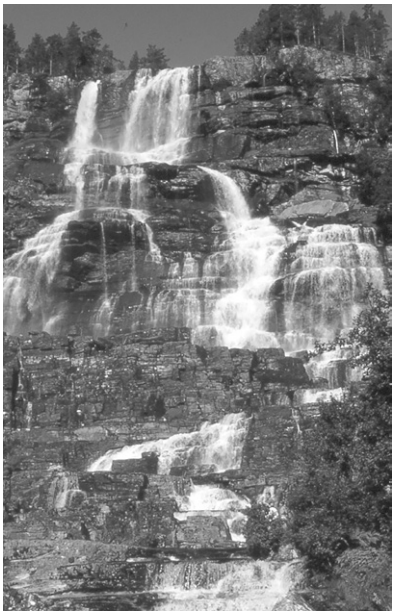


Bild 1.1 Foto von Wolfgang Korn

„Wir entwerfen komplexe technische Systeme“, erklärt ein graumelierter Mittfünfziger der Bauherrin bei ihrem ersten Treffen. Sein Titel „Bürovorsteher“ prädestiniert ihn wohl für den Erstkontakt zu dem vermeintlich kleinen Fisch. Von ihm und einer deutlich jüngeren Assistentin wurde sie ausgiebig nach ihren Wünschen hinsichtlich des geplanten Hauses befragt.

Als sie die Frage nach den Türgriffen des Badezimmerschranks im Obergeschoss nicht spontan beantworten kann, händigt man ihr Formblätter aus, die ausführlich den Change-Management Prozess beschreiben.

Das Team der Wasserfall-Architektur KG legte nach wenigen Wochen einen detaillierten Projektplan vor. Gantt-Charts, Work-Breakdown-Struktur, Meilensteine, alles dabei. Die nächsten Monate verbrachte das Team mit der ausführlichen Dokumentation der Anforderungen sowie dem initialen Entwurf.

Pünktlich zum Ende dieser Phase erhielt die Unternehmerin einen Ordner mit gut 400 Seiten Beschreibung eines Hauses. Nicht ganz das von ihr Gewünschte, weil das Entwicklungsteam aus Zeitgründen einige (der Bauherrin nur wenig zusagende) Annahmen über die Größe mancher Räume und die Farbe einiger Tapeten getroffen hatte. Man habe zuerst überall groben Sand als Bodenbelag geplant, könne das aber später erweitern. Mit etwas Zement und Wasser vermischt, stünden später alle Möglichkeiten offen. Im Rahmen der hierbei erwarteten Änderungen habe das Team vorsorglich die Treppen als Rampe ohne Stufen geplant, um Schubkarren den Weg in die oberen Etagen zu erleichtern. Das Begehren unserer Unternehmerin, doch eine normale Treppe einzubauen, wurde dem Change-Management übergeben.

Die nun folgende Erstellungsphase (die Firma verwendete hierfür den Begriff „Implementierungsphase“) beendete das Team in 13 statt der geplanten acht Monate. Die fünf Monate Zeitverzug seien durch widrige Umstände hervorgerufen, wie ein Firmensprecher auf Nachfrage erklärte. In Wirklichkeit hatte ein Junior-Planning-Consultant es versäumt, einen Zufahrtsweg für Baufahrzeuge zu planen – das bereits fertiggestellte Gartenhaus musste wieder abgerissen werden, um eine passende Baustraße anlegen zu können.

Ansonsten hatte das Implementierungsteam einige kleine Schwächen des Entwurfs optimiert. So hatte das Haus statt Treppe nun einen Lastenaufzug, weil sich die ursprünglich geplante Rampe für Schubkarren als zu steil erwies. Das Change-Management verkündete stolz, man habe bereits erste Schritte zur Anpassung des Sandbodens unternommen: Im ganzen Haus seien auf den Sand Teppiche gelegt worden. Leider hatte ein Mitglied des Wartungsteams über den Teppich dann, in sklavischer Befolgung der Planungsvorgaben, Zement und Wasser aufgebracht und mithilfe ausgeklügelt brachialer Methoden zu einer rotgrauen zähen Paste vermischt. Man werde das in der Wartungsphase optimieren, hieß es seitens der Firma.

Die zu diesem Zeitpunkt von den Wasserfall-Architekten ausgestellte Vorabrechnung belief sich auf das Doppelte der ursprünglich angebotenen Bausumme. Diese Kostensteigerung habe die Bauherrin durch ihre verspätet artikulierten Zusatzwünsche selbst zu verantworten.

V&V Architektur GmbH & Co. KG

Die V&V Architektur GmbH & Co. KG (nachfolgend kurz V&V) hatte sich in den vergangenen Jahren auf Regierungs-, Prunk- und Profanbauten spezialisiert. Mit dem unternehmenseigenen Verfahren, so wird versichert, könne man garantiert jedes Projekt abwickeln. Der von V&V ernannte Projektleiter überraschte unsere Unternehmerin in den ersten Projektwochen mit langen Fragebögen – ohne jeglichen Bezug zum geplanten Haus. Man müsse unbedingt zuerst das Tailoring des Vorgehensmodells durchführen, das Modell exakt dem geplanten Projekt anpassen. Am Ende dieser Phase erhielt sie, in zweifacher Ausfertigung, mehrere Hundert Seiten Dokumentation des geplanten Vorgehens.

Dass ihr Einfamilienhaus darin nicht erwähnt wird, sei völlig normal, unterrichtete sie die Projektleitung. Erst jetzt, in der zweiten Phase, würde das konkrete Objekt geplant, spezifiziert, realisiert, qualitätsgesichert und konfigurationsverwaltet.

Bild 1.2 Foto von Ralf Harder

Der Auftraggeberin wurde zu diesem Zeitpunkt auch das „Direktorat EDV“ der Firma V&V vorgestellt. Entgegen ihrer Annahme befasst sich diese Abteilung nicht mit Datenverarbeitung, sondern mit der „**E***inhaltung* **D***es* **V***orgehensmodells*“.

Nach einigen Monaten Projektlaufzeit stellte unsere Bauherrin im bereits teilweise fertiggestellten Haus störende signalrote Inschriften auf sämtlichen verbauten Teilen fest. Das sei urkundenechte Spezialtinte, die sich garantiert nicht durch Farbe oder Tapete verdecken ließe, erklärte V&V stolz. Für die Qualitätssicherung und das Konfigurationsmanagement seien diese Kennzeichen unbedingt notwendig. Ästhetische Einwände, solche auffälligen Markierungen nicht in Augenhöhe auf Fenstern, Türen und Wänden anzubringen, verwarf die Projektleitung mit Hinweis auf Seite 354, Paragraf 9 Absatz 2 des Vorgehensmodells, in dem Größe, Format, Schrifttyp und Layout dieser Kennzeichen verbindlich definiert seien. Die Bauherrin hätte bereits beim Tailoring widersprechen müssen, nun sei es wirklich zu spät.

Extrem-Architekten AG

Die Extrem-Architekten laden unsere Unternehmerin zu Projektbeginn zu einem Planungsspiel ein. Jeden Raum ihres geplanten EFH soll sie dabei der Wichtigkeit nach mit Gummibärchen bewerten. Die immer nur paarweise auftretenden Architekten versprechen ihr eine erste funktionsfähige Version des Hauses nach nur sechs Wochen. Auf Planungsunterlagen würde man im Zuge der schnellen Entwicklung verzichten.

Bild 1.3 „Gummibär-Tango“ von Klaus Terjung

Zu Beginn der Arbeiten wurde das Team in einer Art Ritual auf die gemeinsame Vision des Hauses eingeschworen. Wie ein Mantra murmelten alle Teammitglieder ständig mit seltsam gutturaler Betonung die Silben „Einfa-Milien-Haus“, was sich nach einiger Zeit zu „Ei-Mi-Ha“ abschliff. Mehrere Außenstehende wollen gehört haben, das Team baue einen bewohnbaren Eimer. Sie stellten eine überdimensionale Tafel am Rande des Baugeländes auf. Jeder durfte darauf Verbesserungsvorschläge oder Änderungen eintragen. Dies gehöre zu einem Grundprinzip der Firma: „Kollektives geistiges Eigentum: Planung und Entwurf gehören allen.“

Nach exakt sechs Wochen laden die Extrem-Architekten die Unternehmerin zur Besichtigung der ersten funktionsfähigen Version ein. Wieder treten ihr zwei Architekten entgegen, jedoch erkennt sie nur einen davon aus dem Planungsspiel wieder. Der andere arbeitet jetzt bei den Gärtnern. Der ursprüngliche andere Gärtner hilft dem Elektriker, ein Heizungsbauer entwickelt dafür die Statik mit. Auf diese Weise verbreite sich das Projektwissen im Team, erläutern beide Architekten eifrig.

Man präsentiert ihr einen Wohnwagen. Ihren Hinweis auf fehlende Küche, Keller und Dachgeschoss nehmen die Extrem-Architekten mit großem Interesse auf (ohne ihn jedoch schriftlich zu fixieren).

Weitere sechs Wochen später hat das Team eine riesige Grube als Keller ausgehoben und den Wohnwagen auf Holzbohlen provisorisch darüber befestigt. Das Kellerfundament haben ein Zimmermann und ein Statiker gegossen. Leider blieb der Beton zu flüssig. Geeignete Tests seien aber bereits entwickelt, dieser Fehler käme garantiert nie wieder vor.

Mehrere weitere 6-Wochen-Zyklen gehen ins Land. Bevor unsere Unternehmerin das Projekt (vorzeitig) für beendet erklärt, findet sie zwar die von ihr gewünschte Küche, leider jedoch im Keller. Ein Refactoring dieses Problems sei nicht effektiv, erklärte man ihr. Dafür habe man im Dach einen Teil der Wohnwagenküche verbaut, sodass insgesamt die Zahl der Küchen-Gummibären erreicht worden sei.

Das immer noch flüssige Kellerfundament hat eines der Teams bewogen, auf die Seitenwände des Hauses auf Dauer zu verzichten, um die Lüftung des Kellers sicherzustellen. Im Übrigen besitzt das Haus nur ein Geschoss, das aktuelle Statik-Team (bestehend aus Zimmermann und Gärtner) hat dafür die Garage in drei Kinderzimmer unterteilt.

Weil das Team nach eigenen Aussagen auf die lästige und schwergewichtige Dokumentation verzichtet hatte, waren auch keine Aufzeichnungen der ursprünglichen Planung mehr erhalten.

Im Nachhinein beriefen sich alle Projektteams auf ihren Erfolg. Niemand hatte bemerkt, dass die Bauherrin keines der „implementierten“ Häuser wirklich akzeptierte.

Chaos nur am Bau?

Keineswegs! Ähnlichkeiten mit bekannten Vorgehensweisen bei der Softwareentwicklung sind ausdrücklich gewollt, denn nicht nur beim Hausbau herrscht Chaos.[1] Auch andere In-

[1] Viele Grüße nach Berlin. Ähnlichkeiten mit gescheiterten Flughafenprojekten sind rein zufällig.

genieurdisziplinen erleben *turbulente Situationen*, obwohl der Maschinenbau über mehr als 200 Jahre Erfahrung verfügt. In der Softwarebranche geht es mindestens ebenso schlimm zu.

Der regelmäßige Chaos-Report der Standish-Group zeigt eine seit Jahren gleichbleibende Tendenz: Über 30 % aller Softwareprojekte werden (erfolglos) vorzeitig beendet, in über 50 % aller Softwareprojekte kommt es zu drastischen Kosten- oder Terminüberschreitungen.[2]

Softwarearchitektur allein kann diese Probleme nicht lösen. Stakeholder mit klaren Zielvorstellungen, ein motiviertes Entwicklungsteam und ein effektives, flexibles und (hoffentlich) agiles Management bilden wichtige Voraussetzungen für erfolgreiche Softwareentwicklung[3].

1.1 Softwarearchitekt:innen

Liebe Softwarearchitektinnen, ich meine überall auch Sie, obwohl ich im weiteren Buch meist die maskuline (weil kürzere) Form „Softwarearchitekt“ verwende. Ich meine grundsätzlich Softwarearchitektinnen und Softwarearchitekten –verwende aber nur sporadisch die längere Form Softwarearchitekt:innen. Das Gleiche gilt für Auftraggeber:innen, Anwender:innen und andere Rollen.

Danke für Ihr Verständnis!

Der Rolle „Softwarearchitektur“ kommt in IT- oder Softwareprojekten besondere Bedeutung zu:

Softwarearchitekten bilden die Schnittstelle zwischen Analyse, Entwurf, Implementierung, Management und Betrieb von Software.

Diese verantwortungsvolle Schlüsselrolle bleibt in vielen Projekten oft unbesetzt oder wird nicht angemessen ausgefüllt. Architekt:innen sollten sicherstellen, dass Anforderungen der Stakeholder einerseits umsetzbar sind und andererseits auch umgesetzt werden.

Langfristigkeit

Softwarearchitekt:innen denken langfristig – auf die gesamte Lebensdauer von IT-Systemen bezogen. Sie ermöglichen kurzfristige Änderungen, sichern gleichzeitig die Langlebigkeit und Nachhaltigkeit von Software. Sie verfolgen konzeptionelle Integrität (auch genannt *Konsistenz*): Die gesamte Konstruktion von Software sollte einem einheitlichen Stil folgen. Insbesondere sollten *ähnliche Aufgabenstellungen in Systemen ähnlich gelöst* werden. Dies erleichtert das Verständnis und die langfristige Weiterentwicklung.

---

[2] Quelle: The Standish Group Chaos Report. Erhältlich unter *https://standishgroup.com*

[3] Eine gute Voraussetzung für Projekterfolg ist es, im Team die Eigenschaften **Kompetenz, Energie** und **Verantwortung** zu bündeln – danke an Dierk König für diese Formulierung.

Konzeptionelle Integrität

Das Buch gibt aktiven und angehenden Softwarearchitekt:innen praktische Ratschläge und Hilfsmittel, diese komplexen Aufgaben effektiver zu erfüllen. Es unterbreitet konkrete Vorschläge, wie Sie bei Softwarearchitektur in der Praxis vorgehen können. Auch wenn Sie in anderen Funktionen in Softwareprojekten arbeiten, kann dieses Buch Ihnen helfen. Sie werden verstehen, welche Bedeutung Architekturen besitzen und wo die Probleme bei Entwurf und Kommunikation von Architekturen liegen.

1.2 Effektiv, agil und pragmatisch

Effektivität, Agilität und Pragmatismus prägen die Grundhaltung erfolgreicher Entwicklungsteams, insbesondere der Softwarearchitektur.

Agilität ist notwendig

Software wird in vielen Projekten immer noch als starres, unveränderliches Produkt betrachtet, obwohl Anwender und Auftraggeber laut nach hochgradig flexiblen Lösungen rufen. In der Praxis ähnelt die Softwareentwicklung leider oftmals eher dem Brückenbau: Eine Rheinbrücke bleibt auch in den kommenden Jahren eine Rheinbrücke. Weder verändert sich der Flusslauf noch wird aus einer Eisenbahnbrücke eine Startbahn für Passagierflugzeuge. Für Software stellt sich die Lage ganz anders dar: Hier kann aus einem abteilungsinternen Informationssystem schnell eine global genutzte Internet-E-Business-Lösung entstehen.

Langjährige Untersuchungen ergeben, dass sich 10 bis 25 % der Anforderungen an Software pro Jahr ändern (Quelle: Peter Hruschka). Management, Architektur und Entwicklung müssen sich in Softwareprojekten durch flexible und bedarfsgerechte Vorgehensweisen darauf einstellen. Das Schlüsselwort lautet „Agilität“.

Agilität und flexibles Vorgehen wird in Softwareprojekten an vielen Stellen dringend benötigt:

- Schon das Requirements-Engineering muss flexibel mit Änderungen von Anforderungen umgehen.
- Softwarearchitekturen müssen stabile Grundgerüste bereitstellen, die die Umsetzung neuer und geänderter Anforderungen ermöglichen. In der heutigen Marktsituation müssen solche Änderungen schnell und effektiv erfolgen – oder sie sind wirkungslos.
- Projekt- und Produktmanagement müssen in der Lage sein, während der Erstellung eines Systems flexibel auf neue Anforderungen, neue Technologien oder aktualisierte Produkte zu reagieren. Hier bietet agiles Vorgehen und risikobasiertes Projektmanagement viele Vorteile gegenüber den strikt am Vorgehensmodell orientierten konventionellen Methoden.
- Dokumentation muss sich an spezifischen Projektbedürfnissen orientieren statt an fix vorgegebenen Ergebnistypen. Inhalt ist wichtiger als Form.
- Agilität erfordert allerdings auch hohe Qualifikation und Professionalität der Beteiligten. Wenn Sie in einem agilen Projekt arbeiten, müssen Sie in allen Belangen mitdenken und Verantwortung übernehmen.

Insgesamt zählt in einem Projekt nur das Ergebnis. Selten kümmern sich Anwender- oder Auftraggeber:innen im Nachhinein um die Einhaltung starrer Vorgehensmodelle.

Softwarearchitektur muss in ihrer Funktion als Schnittstelle zwischen den Projektbeteiligten diesen Ansatz der Agilität aufnehmen und in der Praxis umsetzen.

Handeln Sie agil!

Von Peter Hruschka

b-Agile

Agil heißt, beweglich und flexibel sein. Mitdenken statt „Dienst nach Vorschrift“ und Dogma.

Kein Vorgehensmodell passt für alle Arten von Projekten. Eine agile Vorgehensweise beurteilt das jeweilige Risiko der unterschiedlichen Aufgaben bei der Softwareentwicklung und wählt dann die geeigneten Maßnahmen. Folgende Schwerpunkte werden dabei gesetzt:

- offen für Änderungen statt Festhalten an alten Plänen,
- eher ergebnisorientiert als prozessorientiert,
- „miteinander darüber reden“ statt „gegeneinander schreiben“,
- eher Vertrauen als Kontrolle,
- Bottom-up „Best Practices“ austauschen und etablieren statt Top-down-Vorgaben diktieren.

Trotzdem heißt „Agilität“ nicht, Anarchie zuzulassen:

- Agile Vorgehensmodelle haben Ergebnisse, nur unterscheiden sich diese für unterschiedliche Projekte in Anzahl, Tiefgang und Formalismus.
- Agile Entwicklung kennt Prozesse, nur lassen diese mehr Spielraum für Alternativen und Kreativität.
- Agile Methoden setzen auf Verantwortung; es werden nur notwendige Rollen besetzt.
- Agile Methoden basieren auf Feedback und Iterationen. Fordern und geben *(push und pull)* Sie Rückmeldung zu Ergebnissen – nur so können Teams und Ergebnisse besser werden.

Das Risiko entscheidet: Alle Beteiligten aus Management, Anforderungsanalyse, Architektur, Entwicklung und Test überprüfen ständig Risiken und entscheiden in ihrem jeweiligen Umfeld über notwendige Maßnahmen, damit aus Risiken keine Probleme werden.

Dr. Peter Hruschka (hruschka@b-agile.de) ist unabhängiger Trainer und Methodenberater. Er ist Prinzipal der Atlantic Systems Guild, eines internationalen Think Tank von Methodengurus, deren Bücher den State of the Art wesentlich mitgestaltet haben (https://peterhruschka.eu/).

Effektiv = Ziele erreichen

Weil das Begriffspaar „effektiv und effizient“ immer wieder für Missverständnisse sorgt, möchte ich die Bedeutung beider Wörter hier kurz gegenüberstellen.

Effizient = hoher Wirkungsgrad

Eine Lexikondefinition des Begriffs „Effizienz“ lautet „Wirkungsgrad“, also das Verhältnis von Aufwand zu Ertrag. Wenn Sie Aufgaben effizient erledigen, dann arbeiten Sie also mit hohem Wirkungsgrad. Sie investieren für den gewünschten Ertrag einen minimalen Aufwand. Spitzensportler etwa vermeiden in ihren Disziplinen überflüssige Bewegungen oder Aktionen, was in hochgradig effizientem Ausführen der jeweiligen Sportart resultiert.

Prägnant ausgedrückt, bedeutet das:

Effizient = Dinge richtig machen

Effektiv = zielorientiert

„Effektiv“ bedeutet zielorientiert. Sie arbeiten effektiv, wenn Sie Dinge erledigen, die zur Erreichung Ihrer konkreten Ziele notwendig sind. Auch für diesen Begriff wieder eine prägnante Definition:

Effektiv = die richtigen Dinge machen

Es ist viel wichtiger, die richtigen Dinge zu erledigen, als irgendwelche Dinge besonders effizient zu tun.

Softwarearchitekten müssen in hohem Maße effektiv arbeiten. Kunden und Auftraggeber bestimmen Ziele, Architekten müssen sicherstellen, dass diese Ziele auch erreicht werden.

Effektiv = agil und angemessen

Effektiv bedeutet auch, angemessen und bedarfsgerecht zu agieren. Auf die Entwicklung von Software angewandt, heißt das, sich permanent an den Bedürfnissen der Kunden und Auftraggeber zu orientieren (und nicht starr an den Buchstaben eines formalen Vorgehensmodells). Ich plädiere in diesem Buch für Agilität in diesem Sinne.

Effektive Softwarearchitektur bedeutet, das (für Kunden und Auftraggeber) richtige System zu konstruieren und zu bauen – mit technisch und organisatorisch angemessenen Mitteln.

1.3 Wer sollte dieses Buch lesen?

Grundsätzlich können alle Stakeholder[4] von diesem Buch profitieren und erhalten Antworten auf zentrale Fragen.

- Softwarearchitektur
  - Was sind die Methoden und Werkzeuge unserer Zunft?
  - Wie gehen Sie beim Entwurf von Architekturen sinnvoll vor?
  - Welche praktisch erprobten Heuristiken und Ratschläge gibt es?
  - Wie meistern Sie externe Einflussfaktoren, die den Entwurf von Architekturen erschweren?
  - Wie können Sie Architekturen pragmatisch und effektiv kommunizieren?
  - Was sind die grundlegenden technischen und querschnittlichen Konzepte?
- Softwareentwickler:innen
  - Wie hängen Architektur und Implementierung zusammen?
  - Wie kann das gesamte Entwicklungsteam bei Entwurf und Pflege der Architektur konstruktiv mitwirken?
  - Welche allgemeinen Prinzipien von Softwarearchitektur und -entwurf sollten Sie bei der Implementierung unbedingt befolgen?
- Alle Personen, die sich auf die Prüfung zum „Certified Professional for Software Architecture – Foundation Level“ (CPSA-F) des iSAQB e. V. vorbereiten
- Technische Manager:innen
  - Warum sollen Sie für Architektur Geld ausgeben? Warum ist Softwarearchitektur wichtig?
- Product-Owner, Scrum-Master, Projektleiter:innen
  - Was genau bewirkt Architektur in IT-Projekten?
  - Welche Aufgaben erfüllt Softwarearchitektur?
  - Welche Bedeutung hat die Dokumentation von Architekturen („Was bedeuten all diese Symbole?“)?
  - Welche grundlegenden Lösungsansätze gibt es für Architekturen?
- Business-Analysts und Requirements-Engineers
  - Was bedeuten all diese Begriffe, die das Entwicklungsteam und die Architekten ständig verwenden?
  - Wie können Sie bei der Architekturentwicklung konstruktiv beitragen?

[4] Im Verlauf des Buchs benutze ich den Begriff Stakeholder für Personen oder Organisationen, die bei der Erstellung des Systems mitwirken, es beeinflussen oder am entwickelten System ein fachliches, technisches oder kommerzielles Interesse haben.

1.4 Wegweiser durch das Buch

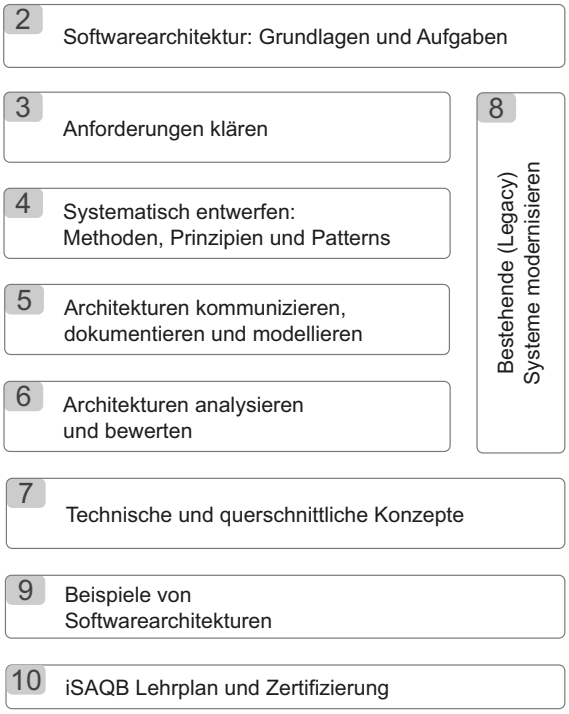


Bild 1.4
Wegweiser

Kapitel 2 klärt die Grundbegriffe rund um Architektur. Es beantwortet die Fragen nach dem Was, Warum und Wer von Softwarearchitekturen. Damit legt es das Fundament für eine systematische und gleichzeitig flexible Vorgehensweise.

Kapitel 3 widmet sich der Aufgabe, Anforderungen systematisch zu klären.

Kapitel 4 gibt Ihnen methodische Werkzeuge für zielgerichtete Entwurfsentscheidungen an die Hand. Sie lernen hier die Grundlagen des Entwurfs, geltende Prinzipien und erprobte Heuristiken. In diesem Teil bekommen Sie eine kompakte Einführung in *Domain-Driven Design*, *Quality-Driven Architecture* sowie in die wesentlichen Architektursichten. Neben diesen Methoden zur Strukturierung Ihrer Systeme und Bausteine lernen Sie hier auch einige wichtige Patterns kennen.

arc42 Template

Kapitel 5 beschreibt, wie Sie Ihre Softwarearchitekturen kommunizieren und dokumentieren können. Außerdem finden Sie hier einige Hinweise für gute Architekturdokumentation sowie das bekannte Architekturtemplate arc42. Zusätzlich lernen Sie einige Grundlagen von UML und dem C4-Ansatz kennen.

Kapitel 6 zeigt, wie Sie methodisch Architekturen analysieren und bewerten können, insbesondere hinsichtlich der geforderten Qualitätseigenschaften.

(technische) Konzepte

Kapitel 7 enthält einen Katalog häufig benötigter technischer Konzepte. Hierzu zählen Persistenz (Datenspeicherung), Integration,

Verteilung, Kommunikation, Sicherheit, grafische Benutzeroberflächen, Ausnahme- und Fehlerbehandlung sowie Management von Geschäftsregeln.

Modernisierung und Evolution

Kapitel 8 beleuchtet Änderung, Modernisierung und Evolution von Software – womit Entwicklungsteams vermutlich 80 % ihrer beruflichen Zeit verbringen.

Beispiele von Architekturen

Kapitel 9 enthält Beispiele von Softwarearchitekturen, beschrieben nach der Strukturvorlage arc42 (die Sie in Kapitel 4 kennengelernt haben).

Vorbereitung auf CPSA-F

In **Kapitel 10** finden Sie einige Hinweise zur Vorbereitung auf die iSAQB Foundation Level Zertifizierung CPSA-F

1.5 Webseite zum Buch

Website: *https://www.esabuch.de*

Auf der Website *https://www.esabuch.de* finden Sie Informationen, die aus Platzgründen ins Internet weichen mussten. Dazu gehören aktuelle Literaturhinweise und Links sowie sonstige Hilfsmittel für Softwarearchitekten zum Download.

https://www.arc42.de

Sie können unter *https://www.arc42.de* den aktuellen Stand des praxisnahen und umfassend erprobten Architekturtemplates herunterladen – hilfreich für Entwurf, Entwicklung und Dokumentation von Softwarearchitekturen.

1.6 Weiterführende Literatur

[DeMarco+07] beschreiben (äußerst humorvoll und gnadenlos wahr) mehr als 80 hilfreiche „Verhaltensmuster“ aus IT-Projekten – die meiner Meinung nach alle ITler kennen sollten.

Ebenfalls in Form informeller Muster nehmen Peter Hruschka und ich im [Knigge] diverse typische positive und negative Verhaltensweisen rund um Softwarearchitektur ins Visier.

Falls Sie Softwarearchitektur an realen Beispielen „erleben“ möchten (und Ihnen Kapitel 9 nicht genügt) – unter *https://leanpub.com/arc42byexample* finden Sie weitere Beispiele (u. a. eine Schach-Engine und ein Portal für Radsport).

Als englische Literatur lege ich Ihnen [Keeling], [Ford+17] und auch [Bass+21] nahe. Falls Sie verteilte Systeme (etwa: Microservices, Self-Contained-Systems oder generell Client/Server) bauen, diskutieren [Ford+21] viele praxisrelevante Fragestellungen.

1.7 Danksagung

Danke an meine Traumfrau, Cheffe Uli, für Engelsgeduld, Motivation und tolle gemeinsame Zeit auf Bergen, Rädern, Yogamatten und Sofas sowie in Gyms. Du sorgst für mein glückliches Leben!

Seit über 25 Jahren habe ich ungemein von den Diskussionen und Arbeiten mit Peter Hruschka rund um Methodik der Softwareentwicklung sowie arc42 profitiert. Danke, dass ich trotz aller inhaltlicher Differenzen immer noch Dein Freund sein darf!

Danke an Phillip Ghadir, Tobias Hahn, Dr. Simon Harrer, Alexander Heusingfeld, Wolfgang Korn, Stefan Tilkov und Oliver Wolf für ihre Beiträge.

Danke an das großartige Team der INNOQ – ihr seid die Besten!

Lynn und Per, für die wirklichen Prioritäten.

Danke an meine zahlreichen Reviewer – namentlich nenne ich euch auf *https://www.esabuch.de*. Ihr habt mich immer wieder auf Fehler und Ungenauigkeiten hingewiesen und das Buch über die Jahre damit immer besser gemacht.

Danke an Dr. Alexander Lorz für erhellende Diskussionen um die Grundlagen unserer Zunft.

Danke auch an die vielen ehrenamtlichen Mitglieder des iSAQB e. V., insbesondere aus der Foundation-Level Working Group, für konstruktive Diskussionen zu grundlegenden Fragen der Softwarearchitektur.

Zu guter Letzt vielen Dank an meine Kunden, dass ich in Ihren/euren Projekten so viel über Softwarearchitekturen erfahren und erleben durfte.

In Gedenken an den großartigen Stefan Tilkov – der viel zu früh gegangen ist. Du warst Vorbild, Mentor, Moderator und Möglichmacher. Du fehlst mir!

2 Softwarearchitektur: Grundlagen und Aufgaben

Architecture is about people.

Norman Foster

Fragen, die dieses Kapitel beantwortet:

- Warum brauchen wir Softwarearchitektur?
- Was gehört zu Softwarearchitektur? Welche Arten von Entscheidungen müssen wir treffen?
- Welche Aufgaben muss Softwarearchitektur lösen?
- Wer könnte diese Aufgaben übernehmen (Rolle von Softwarearchitekt:innen)?

Dieses Kapitel beantwortet die Frage „Was ist Softwarearchitektur?“. Es definiert den Begriff und gibt einen Überblick über die notwendigen Aufgaben und die Rolle von Softwarearchitekt:innen in IT-Projekten.

Starten wir mit ein paar gewichtigen Gründen, warum wir uns überhaupt mit Softwarearchitektur beschäftigen sollten und warum *coole Programmierung* alleine in vielen Fällen nicht genügt:

Darum Softwarearchitektur!

- Architektur soll helfen, die wesentlichen Ziele unserer maßgeblichen Stakeholder zu erreichen. Insbesondere bei kritischen oder gar widersprüchlichen Anforderungen können wir das nicht allein durch geschickte Programmierung erreichen.
- Architektur sorgt dafür, dass wichtige oder kritische Qualitätsanforderungen erfüllt werden, wie Performance, Skalierbarkeit, Flexibilität, Robustheit, Datensicherheit.
- Architektur hilft, das System mit angemessenem Aufwand an Personen und technischen Ressourcen zu entwickeln und zu betreiben (Stichwort: Entwicklungseffizienz). Dabei sorgt Architektur auch für die notwendige Time-to-Market.
- Konzeptionelle Integrität (Konsistenz): Architektur sorgt dafür, dass ähnliche Aufgaben im System durchgängig ähnlich gelöst werden. Das wiederum wirkt sich positiv auf Verständlichkeit und Wartbarkeit von Systemen aus.
- Architektur macht grundlegende Entscheidungen verständlich, bezogen auf Komponenten, Schnittstellen oder die Auswahl von Technologien oder Lösungskonzepten.

2.1 Was ist Softwarearchitektur?

Von den vielen Definitionen der einschlägigen Literatur stelle ich Ihnen diejenige des IEEE-Standards 1471[1] vor:

Softwarearchitektur: Die grundsätzliche Organisation eines **Systems**, verkörpert durch dessen **Komponenten**, deren **Beziehung** zueinander und zur Umgebung sowie die **Prinzipien**, die für seinen Entwurf und seine Evolution gelten.

System, Komponenten, Beziehungen, Prinzipien – diese Begriffe sind Ihnen umgangssprachlich geläufig, im Zusammenhang mit Softwarearchitektur bedürfen sie aber der Erläuterung. Sie wissen schon, ein gemeinsames Verständnis von Begriffen bildet die Grundlage des gegenseitigen Verstehens. Also legen wir los.

2.1.1 System

System: Gruppe von Elementen, die ein einheitliches Ganzes bilden

Das Merriam-Webster Wörterbuch[2] definiert ein System als *„eine regelmäßig interagierende oder voneinander abhängige Gruppe von Elementen, die ein einheitliches Ganzes bilden“*. Diese Definition ist so abstrakt, dass sie jedes Konglomerat beliebiger Elemente abdeckt, die entweder interagieren oder einfach nur zusammengestöpselt sind. Deswegen einige Beispiele für Systeme:

- das einzelne Softwaresystem, an dem wir arbeiten oder das wir benutzen,
- die Mischung aus Hardware und Software eines komplexen Geräts, beispielsweise ein Steuergerät in einem Auto oder auch ein Smartphone,
- eine Menge von Servern und Netzwerken, die wir manchmal als *Cloud* bezeichnen,
- der riesige multinationale Konzern, dem sowohl Hard- als auch Software gehören, und so weiter.

Alternativ (anstatt System) werden die Begriffe *Anwendung* oder *Programm* verwendet. Jedes System ist in eine Umgebung eingebettet und interagiert mit einem oder mehreren anderen Systemen (siehe „Umgebung“, weiter unten).

Um Missverständnisse bei diesem zentralen Begriff zu vermeiden, sollten Sie also genau klären, was bei Ihrem *System* der Umfang (engl. *scope*) sein soll. Definieren Sie möglichst genau, was zu Ihrem System gehören soll und was nicht.

Aber kommen wir zurück zur Definition von Softwarearchitektur und klären noch ein paar weitere Begriffe.

---

[1] Dieser Standard wurde mittlerweile durch ISO/IEC 42010 abgelöst, die ursprüngliche Begriffsdefinition finde ich aber immer noch gut!

[2] *https://www.merriam-webster.com/dictionary/system*

2.1.2 Komponenten

Komponenten: Bestandteile größerer Einheiten

Schon wieder so ein Begriff aus der Alltagssprache, für den wir eine Definition brauchen. Wikipedia[3] definiert Komponenten als „ Komponente (von lateinisch *componens*, das Zusammensetzende) bezeichnet allgemein die Bestandteile größerer Einheiten".

In Software finden sich verschiedenste strukturelle („zusammensetzende") Einheiten: Applikationen, Subsysteme, Services, Module, Klassen, Funktionen. Ganz wichtig: Komponenten können auch zugeliefert werden, also 3rd-Party-Bestandteile, Frameworks oder auch Services sein.

Bezogen auf die Softwareentwicklung verwende ich für Komponenten den allgemeineren Begriff *Baustein*, konform zur Begriffswelt[4] von arc42. Komponenten bestehen meistens aus Quellcode einer Programmiersprache, können aber auch andere Artefakte sein, beispielsweise Konfigurationen. Das gesamte *System* (Abschnitt 2.1.1) besteht aus mehreren solcher Komponenten.

Aus der Perspektive von Hardware, Infrastruktur und Betrieb gehören auch physische Computer, Boards, Microcontroller, Server, Container, Firewalls und andere (echte oder virtuelle) Hardware-Einheiten zu Komponenten.

In typischen Architekturdiagrammen finden wir Komponenten als *Kästen* (auf schweizerdeutsch charmant als *Böxli* bezeichnet).

2.1.3 Beziehungen

Beziehungen: Abhängigkeiten, Verbindungen, Schnittstellen

Schnittstellen, Verbindungen, Abhängigkeiten, Assoziationen: viele Bezeichnungen für dasselbe Phänomen. Komponenten müssen mit anderen Komponenten interagieren, sonst wäre keine Trennung von Belangen (Aufteilung der Verantwortung) möglich. Jedoch treten bei Schnittstellen eine Reihe möglicher Probleme auf:

- Das Entwerfen guter Schnittstellen ist schwierig,
- Die an einer Schnittstelle beteiligten Komponenten oder deren Entwicklungsteams haben manchmal unterschiedliche Bedürfnisse oder Vorstellungen,
- Missverständnisse bei Schnittstellen gelten als Ursache vieler Probleme in Softwaresystemen.

Wie schon bei den Komponenten gibt es auch bei den Beziehungen oder Schnittstellen eine Entsprechung bei Infrastruktur und Hardware: Dort bilden die Beziehungen die physischen Verbindungen zwischen Infrastrukturkomponenten, häufig realisiert durch Netzwerke oder Busse.

Sie haben es sich bestimmt schon gedacht: Beziehungen stellen wir in Diagrammen üblicherweise als Pfeile dar.

---

[3] *https://de.wikipedia.org/wiki/Komponente*
[4] *https://www.arc42.de/method#strukturen-entwerfen*

2.1.4 Umgebung

Jedes System existiert in einer Umgebung (auch genannt *Kontext*): Daten, Events oder Kommandos gelangen aus dieser Umgebung in das System hinein. Andererseits fließen auch Ergebnisse in diese Umgebung zurück.

Diese Außenbeziehungen nennen wir *externe Schnittstellen* und es gibt zwei Ausprägungen:

- Den einen Teil der Umgebung bilden *andere IT-Systeme*, Software oder auch Hardware.
- Den anderen Teil der Umgebung bilden *Akteure*, die das System verwenden, bedienen, konfigurieren oder auf andere Weise damit interagieren.

Als Softwarearchitekt:in müssen Sie die Umgebung Ihrer Systeme kennen und beachten.

Die Kontextsicht (auch genannt Kontextabgrenzung) auf Software hebt die Bedeutung dieser externen Schnittstellen hervor (siehe Kapitel 4). Das folgende Beispiel zeigt das System (aka: die Anwendung) im Zentrum und die Nachbarsysteme schattiert am Rand des Bilds.

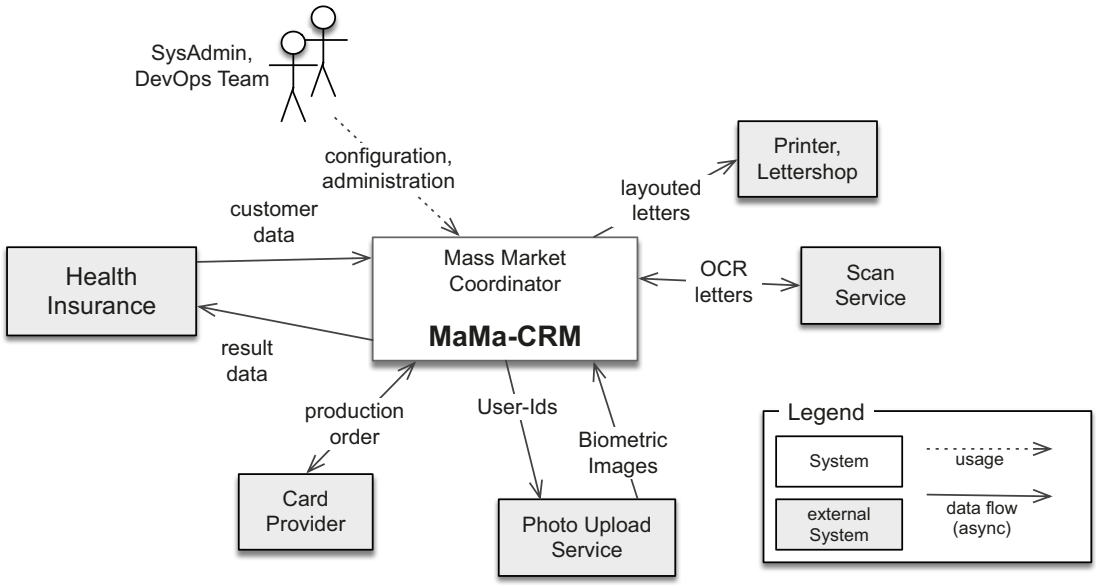


Bild 2.1 Beispiel einer Kontextabgrenzung (Sicht auf die Umgebung)

2.1.5 Komponenten + Beziehungen = Strukturen

Struktur: innere Ordnung eines Systems

Obwohl der Begriff *Struktur* in der praktischen Definition von Softwarearchitektur (siehe Anfang dieses Abschnitts) gar nicht vorkommt, möchte ich dennoch darauf eingehen. Wie immer zuerst mal eine Begriffsklärung:

Struktur (von lateinisch *strūctūra* „Bauart“ bzw. „Zusammenfügung“) bezeichnet die innere Ordnung eines Systems, also dessen Aufbau aus Komponenten und deren Beziehungen.

Wir sprechen bei der Architekturarbeit von *Strukturentscheidungen*, wenn wir die Zerlegung oder den Schnitt des Systems in Komponenten und deren Beziehungen meinen. Mit einer solchen Zerlegung schaffen wir eine *innere Ordnung*.

Sie kennen Strukturen garantiert aus den typischen Architekturdiagrammen, also Kästchen (Komponenten oder Bausteine) mit ihren Beziehungen (Pfeile). Genau genommen gehören zur Struktur auch die Beziehungen zur Umgebung, aber das haben Sie sich sicherlich schon gedacht.

Schon mal als Teaser: Das Duo aus Komponenten und Beziehungen (also die *Struktur*) wird im weiteren Verlauf des Buchs (und Ihrer praktischen Architekturarbeit) noch besondere Bedeutung erhalten. Insbesondere prophezeie ich Ihnen, dass Sie damit in der Praxis eine Menge Arbeit haben werden.

2.1.6 Prinzipien (synonym: Konzepte)

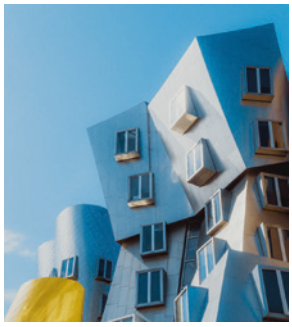
Konzepte, Prinzipien, Technologien, Regeln, Patterns oder andere querschnittlich geltende Entscheidungen

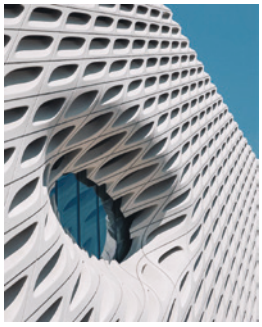
Prinzipien sind Regeln, Patterns oder sonstige querschnittliche Entscheidungen, die für mehrere Teile oder sogar das gesamte System gelten. Ich bevorzuge den Begriff Konzept anstelle von Prinzipien.

Hierzu gehören auch Technologieentscheidungen, beispielsweise die Festlegung von Programmiersprachen, Datenbanksystemen oder User-Interface-Technologien. Einige Themen finden Sie in Kapitel 7.

Konzepte bilden die Grundlage für *konzeptuelle Integrität* oder *Konsistenz* und das wiederum stellt eine der wichtigsten positiven[5] Eigenschaften von Softwaresystemen dar. Im Alltag beispielsweise sind innerhalb eines Gebäudes die Tür- und Fenstergriffe einheitlich beschaffen, ebenso die Art der Fensterrahmen oder Lichtschalter. In Bild 2.2 sehen Sie drei verschiedene Konzepte für Fenster, die für jeweils ein System (Gebäude) einheitlich angewendet worden sind. Einheitlich heißt dabei nicht unbedingt, dass alle völlig identisch aussehen, sondern ähnlichen Prinzipien, Mustern oder Regeln folgen.

[Ján Jakub Naništa@Unsplash]





[Meriç Dağlı@Unsplash]

Bild 2.2 Beispiele für Fenster-Konzepte („Prinzipien“) an Gebäuden

[5] Siehe *https://www.innoq.com/de/blog/whats-in-a-name-consistency/*

Ein paar Beispiele für (Überschriften von) Konzepte in Softwaresystemen:

- Speicherung der Anwendungsdaten bevorzugt in PostgreSQL mithilfe von *SpringData**
- Sämtliche grafischen Oberflächen werden mit *quasar.dev*** implementiert.
- Build und Deploy des Systems erfolgt auf GitLab.
- Sämtliche öffentlichen APIs werden vor jedem Major-Release durch einen extern vergebenen Penetration-Test auf mögliche Schwachstellen hin überprüft.

* Siehe *https://spring.io/projects/spring-data/*
** Siehe *https://quasar.dev*

Konzepte bieten einheitliche Lösungen für wiederholt auftretende Probleme oder Aufgaben in Software. Teams können Konzepte als Teil der systemspezifischen Entwicklung entwerfen und implementieren. Allerdings sind Konzepte oftmals wiederverwendbar, Teams können daher Konzepte aus anderen Systemen übernehmen.

Konzepte enthalten oftmals konkrete Technologie- oder Produktentscheidungen (etwa: Spring Boot, Hibernate, .NET-Core, Angular, React, PostgreSQL).

Konzepte können auch das spezifische Vorgehen in der Entwicklung, Deployment oder Test von Systemen betreffen, siehe „Penetration-Test“ im obigen Kasten.

Oft bestimmen spezifische Qualitätsanforderungen (wie Performance, Skalierbarkeit, Robustheit oder Sicherheit) oder Randbedingungen (wie Budget oder gesetzliche Vorgaben) die Auswahl und Ausgestaltung solcher Konzepte.

2.1.7 Entwurf und Evolution

In der Softwarearchitektur werden übergreifende und systemweite Entscheidungen sowohl beim *initialen Entwurf* als auch während der laufenden Entwicklung (der *Evolution*) von Systemen getroffen. In diesem Sinne ist Architektur eine begleitende Tätigkeit. Architekturrelevante Entscheidungen können zu jedem Zeitpunkt anfallen, an dem irgendwelche Änderungen am System, dessen unmittelbarer Umgebung (Abschnitt 2.1.4, Kontext) oder dessen Infrastruktur nötig werden.

Damit haben wir die wesentlichen Begriffe aus der Definition „Softwarearchitektur“ geklärt. Im Wesentlichen geht es darum, relevante Entscheidungen hinsichtlich der folgenden Themen zu treffen:

- Struktur, also Komponenten und deren Beziehungen
- Querschnittliche Konzepte (Prinzipien, Regeln, Patterns), damit verbunden auch
- Technologie, sowohl Software wie auch Hardware

Im folgenden Abschnitt klären wir, auf Basis welcher Grundlagen Sie diese Entscheidungen treffen können, wann Sie das tun sollten und welche Fragestellungen überhaupt architekturrelevant sind.

2.2 Architekturentscheidungen

Zuerst einmal basieren Architekturentscheidungen (natürlich!) auf den Anforderungen an Systeme. Diese wiederum kommen aber aus unterschiedlichen Quellen und erweisen sich in der Praxis oftmals als schwierig.

Nachfolgend finden Sie eine (möglicherweise unvollständige) Sammlung von Parametern, die Sie Architekturentscheidungen zugrunde legen sollten:

- Funktionale Anforderungen: Was soll das System tun, welche Aufgaben soll es lösen?
- Qualitätsanforderungen: Wie gut soll das System diese Aufgaben lösen? Hierzu gehören Themen wie Performance, Sicherheit, Skalierbarkeit, Zuverlässigkeit, Benutzungsfreundlichkeit, Flexibilität, aber auch Time-to-Market und weitere.
- Sind diese Anforderungen spezifisch und präzise bekannt oder noch unsicher? Gibt es bei den Anforderungen eklatante Widersprüche, die Sie nur durch Kompromisse auflösen können?
- Gibt es Vorgaben zur Technologie, etwa hinsichtlich Entwicklungs- und Betriebsumgebungen oder auch der beteiligten Hardware? Müssen bestimmte Sprachen oder Compiler eingesetzt werden? Muss die Software auf vorgegebener Hardware (etwa: spezifische Microcontroller) mit definierten Betriebssystemen ablaufen?
- Welches Team soll das System entwerfen, umsetzen und betreiben? Wie viel Erfahrung besitzen die Beteiligten? Großes oder kleines Team? Hat das Team gemeinsam bereits ähnliche Systeme umgesetzt?
- Welche Risiken bestehen bei der Entwicklung oder dem Betrieb des Systems? Beispiel: Die Software medizinischer Geräte hat oftmals unmittelbaren Einfluss auf Gesundheit oder sogar Menschenleben, daher ist in solchen Fällen erheblich gründlicheres Arbeiten erforderlich.
- Wie viel Geld oder Zeit steht zur Verfügung?
- Welche organisatorischen Randbedingungen gelten? Gibt es Vorgaben zu Entwicklungs- oder Betriebsprozessen? Arbeiten wir in einem homogenen Team vor Ort oder gar in einem international verteilten Team mit Offshore-Anteilen?
- Welche gesetzlichen Vorgaben müssen wir einhalten? Beispiel: die strikten Vorschriften zu Datenschutz und -sicherheit vieler europäischer Länder (im Gegensatz zu z. B. den USA).

Aufgrund der Vielzahl und Themenbreite dieser Einflüsse schlage ich eine Grundregel für Softwarearchitektur vor:

Grundregel für Softwarearchitektur: KDA, Kommt-Drauf-An

Es kommt immer auf die spezifische Situation an: *One size doesn't fit all*.

Weder sind Microservices für alle Projekte geeignet noch löst <Werkzeug X> (ersetzen Sie hier X durch ein beliebiges IT-Werkzeug Ihrer Wahl) sämtliche Probleme. Es gibt in der Software- und Systementwicklung keine universellen Standardlösungen, keine *silver bullet*.

Viele Muster, Vorgehensweisen, methodischen Ansätze oder auch Technologien passen in einigen Bereichen oder Domänen vielleicht häufig, aber in spezifischen Situationen müssen Sie spezifische Entscheidungen treffen.

Es kommt darauf an: auf die Situation, das Team und dessen Erfahrungen, Risiken und Kritikalität des Systems, Geld oder Budget, verfügbare Zeit, Technologien, Infrastruktur, Entwicklungs-, Test- und Betriebsprozesse und andere Faktoren.

Es gehört zur Architekturarbeit, dieses „kommt-drauf-an" in gegebenen Situationen mit konkreten und zur jeweiligen Situation passenden Entscheidungen zu füllen.

Wie Sie Architekturentscheidungen pragmatisch und effektiv beschreiben können, finden Sie in Abschnitt 5.4.7 erklärt.

Architektur benötigt Diplomatie, Politik und Akrobatik

Stellen Sie sich vor, zwei Ihrer maßgeblichen Stakeholder formulieren klar, präzise und verständlich zwei Anforderungen R1 und R2. Ein dritter Stakeholder gibt Ihnen die Hardwareplattform H vor, auch das verständlich und eindeutig.

Architektur benötigt (oft) Kompromisse

Es stellt sich während der Entwicklung heraus, dass R1 auf H zwar problemlos funktioniert, aber den gesamten verfügbaren Speicher der Hardware benötigt. Desgleichen gilt für die Funktion R2. Das hätte ein Anforderungsmanagement definitiv nicht vor der Entwicklung feststellen können.

Willkommen in der Realität.

Jetzt benötigen Sie diplomatisches oder politisches Geschick: Sie müssen Kompromisse zwischen widersprüchlichen oder konkurrierenden Anforderungen finden und mit den Beteiligten aushandeln. Dabei jonglieren Sie mit Komponenten, Schnittstellen, Technologien, aber auch mit Infrastruktur, Betriebssystemen, Middleware und anderem. Vor allem müssen Sie diese Probleme den Beteiligten *erklären* und verständlich machen.

Wann entscheiden? Früh oder spät?

Unter Unsicherheit entscheiden

Am Ende eines erfolgreichen Projekts ist es einfach, Entwurfsentscheidungen zu kritisieren oder zu loben. Zu diesem Zeitpunkt wissen alle Beteiligten über Technologien, Produkte und auch Anforderungen und die Konsequenzen der getroffenen Entscheidungen genau Bescheid.

Mitten im Stress der Entwicklung können Sie solche Dinge zu großen Teilen nur vermuten. Sie verfügen in der Softwarearchitektur oftmals nicht über genügend Informationen, um *optimale* Entscheidungen zu treffen. Damit die Entwicklung weiterlaufen kann, müssen die Beteiligten in solchen Fällen Mut zu (möglicherweise) suboptimalen Entscheidungen unter Unsicherheit aufbringen. Aber wann sollen wir denn überhaupt entscheiden?

Beachten Sie den Unterschied zwischen Mut und Waghalsigkeit:

Mutig bedeutet, manche Risiken bewusst einzugehen, auch gegen den Willen anderer. Softwarearchitekt:innen* benötigen Mut zu unbequemen Entscheidungen, zu potenziellen Konflikten mit anderen Projektbeteiligten, zu frühzeitigem Eingeständnis von früheren Fehlentscheidungen.

Waghalsigkeit hingegen nenne ich schnelle Entscheidungen ohne bewusste Risikoabwägung, ohne Beachtung von Konsequenzen, ohne Feedback von anderen Beteiligten oder ohne Prüfung möglicher Alternativen.

* Etwas später schauen wir uns die Rolle „Softwarearchitektur“ genauer an und diskutieren, ob eine oder mehrere Personen deren Aufgaben übernehmen. Hier schon mal der Spoiler: Meiner Ansicht nach sollten wesentliche Architekturentscheidungen von mehreren Personen, möglicherweise sogar im gesamten Team getroffen werden.

Entwicklungsteams lernen im Laufe der Entwicklung immer mehr dazu, über Anforderungen, die Domäne, genutzte Technologien, Prozesse und Stakeholder. Treffen Sie Entscheidungen später, haben Sie daher in der Regel mehr Wissen und Erfahrung. Andererseits kann eine verzögerte Entscheidung beispielsweise zu Terminproblemen oder Unsicherheit im Team führen.

Für Architekturentscheidungen den LRM finden

Der Fachbegriff dazu lautet *latest responsible moment* (LRM, spätester vernünftiger Moment). Die Wahl dieses Entscheidungszeitpunkts LRM ist relevant. Versuchen Sie, möglichst viel über die Fragestellung zu lernen[6], bevor sie oder das Team entscheiden. Aber: Warten Sie nicht zu lange ☹

Welche Fragestellungen sind relevant für die Architektur?

Architektur soll sich um grundlegende, relevante Fragestellungen kümmern, aber nicht um alle Details. Welche Themen für ein spezifisches System Relevanz besitzen, können wir gemäß der KDA-Regel aus dem vorigen Abschnitt nicht allgemein festlegen, aber folgende Checkliste (nach Olaf Zimmermann[7]) kann helfen:

- Hat die Fragestellung oder Anforderung große finanzielle Auswirkungen oder Risiken für das beteiligte Business?
- Betrifft es besonders wichtige Stakeholder (etwa: Vorstand oder Geschäftsführung)?
- Betrifft es wichtige Laufzeiteigenschaften, wie Performance oder IT-Sicherheit?
- Betrifft es externe Abhängigkeiten, die unvorhersehbares oder unkontrollierbares Verhalten zeigen?
- Betrifft es mehrere Teile des Systems, hat es querschnittliche Auswirkungen?
- Ist es etwas komplett Neues („*first-of-a-kind*“)
- Hat es in früheren Projekten schon für Ärger gesorgt?

Falls eine der Antworten „Ja“ lautet, sollten Sie von Architekturrelevanz ausgehen und die betreffende Frage oder Anforderungen entsprechend wichtig nehmen.

---

[6] [Toth-19] bezeichnet die Zeit zwischen dem Erkennen einer Fragestellung (sprich: der Notwendigkeit einer Entscheidung) und einem guten Zeitpunkt der Entscheidung als *Lernfenster*.

[7] *https://medium.com/olzzio/architectural-significance-test-9ff17a9b4490*

2.3 Die Aufgaben von Softwarearchitekt:innen

Das Leben von Softwarearchitekten besteht aus einer langen und schnellen Abfolge suboptimaler Entwurfsentscheidungen, die meist im Dunkeln getroffen werden. [Kruchten2001]

In diesem Abschnitt lernen Sie die notwendigen Aufgaben und Tätigkeiten der Softwarearchitektur kennen. In Abschnitt 2.1 haben Sie die wesentlichen Bestandteile von Softwarearchitektur kennengelernt: System, Komponenten, Schnittstellen und (querschnittliche) Konzepte:

- Komponenten und deren Schnittstellen, also Beziehungen zueinander, und
- Schnittstellen zur Umgebung, sogenannte *externe Schnittstellen*
- (querschnittliche) Konzepte (syn: Prinzipien) von Entwicklung, Betrieb und Evolution, beispielsweise die Auswahl von Programmiertechnologie, eines Technologiestacks oder die Verwendung definierter Werkzeuge oder Vorgehensweisen bei Entwicklung, Test oder Rollout/Release.

Damit gibt uns die Definition von Softwarearchitektur bereits zwei (Entscheidungs- oder Entwurfs-) Aufgaben vor: einerseits das Entscheiden über Strukturen (Komponenten und Schnittstellen), andererseits das Entscheiden über querschnittliche Konzepte und Technologien.

Aber Vorsicht: „Entwerfen“ macht nur einen Teil unserer Architekturarbeit aus. Bild 2.3 ordnet die beiden genannten Entwurfsaufgaben ein und stellt alle Architekturaufgaben dar.

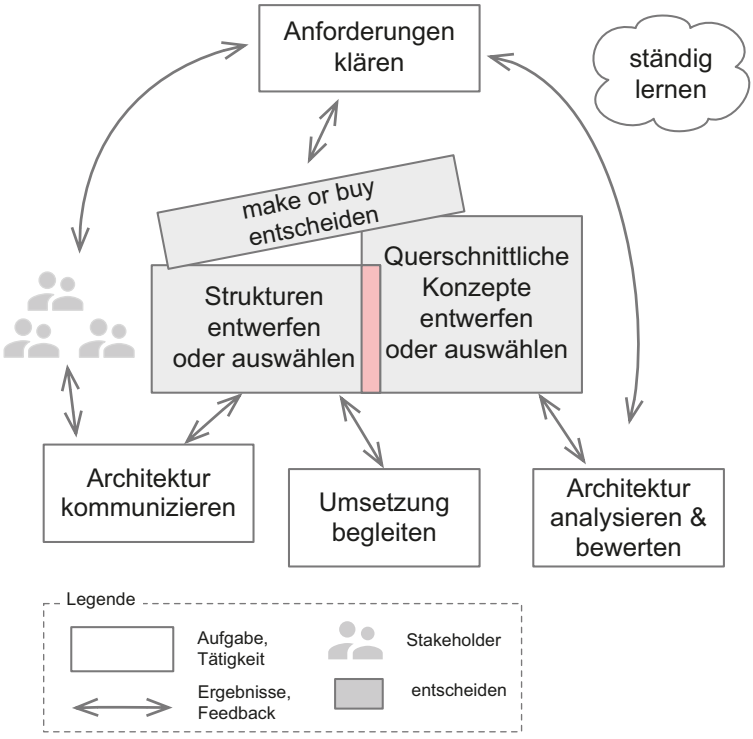


Bild 2.3 Architekturaufgaben

Aufgabe	Bedeutung	Weitere Erklärung
Anforderungen klären	Anforderungen bilden das Fundament angemessener Architekturentscheidungen.	Kapitel 3
Make-or-buy entscheiden	Manchmal ist „kaufen“ geschickter als selbst entwickeln oder implementieren.	Abschnitt 4.1
Strukturen entwerfen	Komponenten und Schnittstellen entwerfen oder passende auswählen	Gesamtes Kapitel 4
Querschnittliche Konzepte entwerfen	Technologien auswählen und deren Einsatz festlegen, über querschnittlich geltende Prinzipien, Regeln und Patterns entscheiden oder passende auswählen	Kapitel 4, Kapitel 7
Architektur kommunizieren	Entscheidungen erklären, andere Stakeholder von der Architektur überzeugen, relevante Sachverhalte dokumentieren	Kapitel 5
Umsetzung begleiten	Wird so programmiert, wie das die Architektur vorgibt oder empfiehlt? Oder hat jemand eine bessere Idee umgesetzt/implementiert?	Abschnitt 2.3.4
Architektur analysieren & bewerten	Systematische Bestandsaufnahme, was läuft in der Architektur gut, wo gibt es Risiken und Probleme?	Kapitel 6
Ständig lernen	Ständig gibt es in der IT Neuerungen, sowohl technischer wie auch methodischer Art. In Ihrer Rolle als Softwarearchitekt:in sollten Sie stets den Überblick über aktuelle Trends und Technologie in dem für Sie relevanten Umfeld behalten.	Sie müssen selbst aktiv werden: Lesen Sie Fachzeitschriften, Blogs und ab-und-zu Fachbücher.

Im weiteren Verlauf des Buchs finden Sie weitergehende Details. Ich habe mich dabei auf die „konstruktiven“ Architekturaufgaben konzentriert und diesen eigene Kapitel gewidmet. Das sind insbesondere die Entwurfsaufgaben (Kapitel 4 und Kapitel 7), die Kommunikation/Dokumentation (Kapitel 4) sowie die Analyse und Bewertung von Architekturen (Kapitel 5).

Zuerst noch einige Anmerkungen zu den Aufgaben aus Bild 2.3.

2.3.1 Anforderungen klären

Als Grundlage Ihrer Entscheidungen sollten Sie eine Vorstellung wichtiger Anforderungen und Eigenschaften des Systems besitzen und folgende Fragen über das System beantwortet haben. Natürlich können Sie diese Klärung iterativ und inkrementell angehen.

Anforderungen klären: Kapitel 3

- Welche Personen oder Gruppen haben Interessen am System (Stakeholder)?
- Was sind die Kernaufgaben des Systems (= funktionale Anforderungen)? Welches sind die wichtigsten Elemente der Fachdomäne? Welche Aufgaben oder Prozesse muss das System unterstützen?
- Welche Qualitätsanforderungen muss das System erfüllen?

- Was sind die (fachlichen und technischen) Nachbarsysteme (externe Schnittstellen, Kontextabgrenzung)?

Unsere Architekturaufgabe ist es primär, diese Anforderungen zu *klären*, und zwar passend zu unserem Vorgehen. Üblicherweise kümmert sich eine andere Rolle (etwa: Requirements-Engineering, Business-Analyse oder Product-Owner) um die Erstellung, Priorisierung und Pflege dieser Anforderungen. Sollten Anforderungen zu schwammig, unklar und widersprüchlich sein oder gar komplett fehlen, müssen Sie in der Architektur handeln statt jammern, also gemeinsam mit Stakeholdern nachbessern oder zumindest über die für Requirements verantwortlichen Stakeholder nachfordern.

Weitere Details finden Sie in Kapitel 3.

2.3.2 Drei Kategorien von Entwurfsentscheidungen

Sowohl Strukturen (= Komponenten + Beziehungen) wie auch querschnittliche Konzepte haben Sie bereits kennen gelernt. Für beide müssen Sie im Rahmen Ihrer Architekturarbeit Entscheidungen treffen, also sowohl Strukturen als auch Konzepte entwerfen.

Ob Sie persönlich diese Entscheidungen treffen oder sich mit anderen Personen oder sogar dem gesamten Entwicklungsteam abstimmen, ist eine komplett andere Frage, die wir in Abschnitt 2.4 ausführlich beantworten werden. Momentan schreibe ich mal unpersönlich „die Architektur“, statt mich auf bestimmte Personen zu beziehen:

1. Durch „Strukturen entwerfen“ entscheidet die Architektur über die Zerlegung (auch genannt: Schnitt) des Systems. Sie bestimmt dabei die Bestandteile (Komponenten, Module, Services, Pakete oder wie auch immer in Ihrer gewählten Technologie die einzelnen Bestandteile eines Gesamtsystems heißen). Ganz wesentlich hierbei sind die Schnittstellen zwischen den einzelnen Bestandteilen sowie zur Umwelt.
2. Durch „Konzepte entwerfen“ entscheidet Architektur beispielsweise über genutzte Technologien und Frameworks. Sie entscheidet über die Art und Weise, wie diese Technologien eingesetzt werden und gibt Prinzipien, Patterns (Muster) und Regeln für architekturrelevante Themen vor.
3. Make-or-Buy entscheiden: In manchen Fällen ist es angemessen, komplett oder teilweise auf bestehende Lösungen, Komponenten oder Systeme zurückzugreifen. Bevor die Architektur also Komponenten und Schnittstellen entwirft, sollten Sie prüfen, ob eine gekaufte (oder Open-Source-) Lösung eventuell günstiger oder geschickter ist als eine selbst implementierte.

Diese Entscheidungsaufgaben stellen den inhaltlichen Kern der Architekturtätigkeit dar. Details finden Sie in Kapitel 4.

2.3.3 Architektur kommunizieren und dokumentieren

Bisher haben Sie Anforderungen geklärt und Entscheidungen getroffen. Wie aber gelangen diese Entscheidungen und deren Erläuterungen zum Entwicklungsteam oder anderen Beteiligten?

Sie müssen diese Entscheidungen *kommunizieren* (in der Regel mündlich) oder *dokumentieren* (üblicherweise schriftlich), damit andere Menschen mit diesen Architekturentscheidungen arbeiten oder darauf mit Feedback reagieren können. Sie erkennen das an den bidirektionalen Pfeilen zwischen den Aktivitäten.

Kommunizieren und dokumentieren: siehe Kapitel 5

Dazu gehören sowohl der Wille als auch die Fähigkeit, technische Sachverhalte für unterschiedliche Stakeholder angemessen aufzubereiten (siehe Kapitel 4), zu präsentieren und zu diskutieren.

Als Architekt:in sollten Sie offen für konstruktive Kritik sein, das ist auch eine Art der Kommunikation. Fordern Sie aktiv Feedback zu Vorschlägen oder Entscheidungen ein.

2.3.4 Umsetzung begleiten: Von Goldstücken und Missverständnissen

Es besteht beim Arbeiten im Team immer das Risiko, dass Menschen sich missverstehen. Dies ist ein menschliches Grundproblem (oder -feature), daher können Sie das nicht ändern. Wenn Sie dem Team etwas erklären, egal ob mündlich oder schriftlich, könnten Personen etwas anderes in diese Worte und Bilder hineininterpretieren als das, was Sie damit eigentlich aussagen wollten.

Solche Missverständnisse haben wir alle in der Realität schon erlebt. Sie müssen in Ihrer Architekturarbeit aktiv etwas dagegen unternehmen: Begleiten Sie die Umsetzung! Prüfen Sie beispielsweise, ob der implementierte Code so beschaffen ist, wie Sie das in der Architektur vorgesehen haben.

Codereviews, Pull-/Merge-Requests oder statische Codeanalyse sind nur einige der methodischen Mittel, die Sie hierfür einsetzen können. Design-Reviews, Pair- oder Mob-Programming[8], Coding-Styleguides, Referenzimplementierungen, Checklisten helfen und noch viele andere.

Umsetzung begleiten hat allerdings noch einen wichtigen zweiten Effekt: Ihre Teamkolleg:innen werden an manchen Stellen schlichtweg auf geeignetere (und somit bessere) Ideen kommen. Solche strukturellen oder technischen Verbesserungen, Vereinfachungen, geschicktere Ansätze oder Ähnliches bezeichne ich als „Goldstücke“ und die sollten Einzug in die Architektur halten.

2.3.5 Architektur analysieren und bewerten

Die IT gehört zu den Branchen mit hohem Anteil an Neuerungen. Selbst fachliche Anforderungen an Systeme unterliegen teilweise heftigem Wandel. Daher sollten Sie ab und zu (etwa halbjährlich oder jährlich) die grundsätzliche Frage stellen, ob Architektur und Lösungsansätze überhaupt noch zum aktuellen Stand der Anforderungen passen.

Ein ergebnisoffenes Review, mindestens aber eine qualitative Architekturanalyse, kann hier hilfreiche Impulse für zukünftige Verbesserungen geben. Details zu dieser Architekturaufgabe finden Sie in Kapitel 6.

[8] Siehe *https://www.remotemobprogramming.org/*, sehr empfehlenswert.