HANSER

Leseprobe

zu

Effektive Softwarearchitekturen

von Gernot Starke

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

Weitere Informationen und Bestellungen unter
https://www.hanser-kundencenter.de/fachbuch/artikel/9783446476721
sowie im Buchhandel

Inhalt

1 Einleitung 1
1.1 Softwarearchitekt:innen 5
1.2 Effektiv, agil und pragmatisch 6
1.3 Wer sollte dieses Buch lesen? 9
1.4 Wegweiser durch das Buch 10
1.5 Webseite zum Buch 11
1.6 Weiterführende Literatur 11
1.7 Danksagung 12

2 Softwarearchitektur: Grundlagen und Aufgaben 13
2.1 Was ist Softwarearchitektur? 14
2.1.1 System 14
2.1.2 Komponenten 15
2.1.3 Beziehungen 15
2.1.4 Umgebung 16
2.1.5 Komponenten + Beziehungen = Strukturen 16
2.1.6 Prinzipien (synonym: Konzepte) 17
2.1.7 Entwurf und Evolution 18
2.2 Architekturentscheidungen 19
2.3 Die Aufgaben von Softwarearchitekt:innen 22
2.3.1 Anforderungen klären 23
2.3.2 Drei Kategorien von Entwurfsentscheidungen 24
2.3.3 Architektur kommunizieren und dokumentieren 24
2.3.4 Umsetzung begleiten: Von Goldstücken und Missverständnissen 25
2.3.5 Architektur analysieren und bewerten 25
2.4 Rolle von Softwarearchitekt:innen: Wer macht's? 26
2.4.1 Monarchie 27
2.4.2 Architekt:in im Team 28
2.4.3 Architekturagent:innen 29
2.4.4 Demokratie oder: Team-Architektur 31
2.5 Architekturen entstehen (meist) iterativ 32
2.6 Weiterführende Literatur 34

3 Anforderungen klären 35
3.1 Was ist Kernaufgabe oder Ziel des Systems? 35
3.2 Relevante Stakeholder ermitteln 36
3.3 Welche Kategorie von System? 37
3.4 Fachdomäne klären 38
3.5 Qualitätsanforderungen klären 39
3.6 Externe Nachbarsysteme: Kontextabgrenzung 44
3.7 Einflussfaktoren und Randbedingungen ermitteln 45

4 Entwurf: Grundlagen, Methoden und Muster 47
4.1 Grundlagen, Prinzipien und Heuristiken 48
4.1.1 Grundlagen des Entwurfs 48
4.1.2 Prinzipien 51
4.1.2.1 Lose (geringe) Kopplung 51
4.1.2.2 Hohe Kohäsion 53
4.1.2.3 Trenne Verantwortlichkeiten/Belange 54
4.1.2.4 Modularisierung 55
4.1.2.5 Abstraktion, Kapselung und das Geheimnisprinzip 55
4.1.2.6 Hohe Konsistenz 55
4.1.2.7 Keine zyklischen Abhängigkeiten 56
4.1.2.8 SOLID-Prinzipien des objektorientierten Entwurfs 57
4.1.3 Heuristiken 62
4.2 Entwurfsmethoden 66
4.2.1 Domain-Driven Design (Entwurf nach Fachlichkeit) 67
4.2.2 Quality-Driven Software Architecture 72
4.2.3 Top-down und Bottom-up 79
4.2.4 Sichtenbasierter Entwurf 80
4.2.4.1 Sichten in der Softwarearchitektur 81
4.2.4.2 Entwurf der Kontextabgrenzung 82
4.2.4.3 Entwurf der Bausteinsicht 83
4.2.4.4 Entwurf der Laufzeitsicht 83
4.2.4.5 Entwurf der Verteilungssicht 84
4.3 Schnittstellen entwerfen 84
4.3.1 Anforderungen an Schnittstellen 86
4.3.2 Worauf müssen Sie achten? 86
4.3.3 Tipps zum Entwurf von Schnittstellen 87
4.4 Architekturmuster (Patterns) 88
4.4.1 Schichten (Layer) 88
4.4.2 Ports-und-Adapter 91
4.4.3 Client-Server 94
4.4.4 Microservices 94
4.4.5 Pipes und Filter 102
4.4.6 Batch-Pattern 104

4.4.7 Repository . . . 105
4.4.8 Blackboard . . . 106
4.4.9 Command Query Responsibility Segregation (CQRS) . . . 107
4.4.10 Broker . . . 109
4.4.11 Peer-to-Peer . . . 110
4.4.12 Ereignisbasierte Systeme – Event Systems . . . 111
4.4.12.1 Ungepufferte Event Systems . . . 112
4.4.12.2 Message- oder Event-Queues . . . 112
4.4.12.3 Message-Service . . . 113
4.4.13 Model-View-Controller . . . 114
4.4.14 Presentation Model . . . 115
4.4.15 REST-Architektur . . . 117
4.4.16 Adapter . . . 119
4.4.17 Stellvertreter (Proxy) . . . 120
4.4.18 Fassade . . . 121
4.4.19 Beobachter (Observer) . . . 122
4.5 Weiterführende Literatur . . . 123

5 Architekturen kommunizieren, dokumentieren und modellieren . 125
5.1 Warum kommunizieren und dokumentieren . . . 126
5.2 Anforderungen an Architekturdokumentation . . . 128
5.3 Effektiv dokumentieren . . . 129
5.3.1 Tipps für bessere Architekturdiagramme . . . 131
5.4 Bestandteile von Architekturdokumentation . . . 138
5.4.1 Kontextabgrenzung: Vogelperspektive . . . 138
5.4.2 Bausteinsicht: Code-im-Großen . . . 140
5.4.3 Schnittstellen: Die Brücken zwischen Welten . . . 143
5.4.4 Laufzeitsicht: Was geschieht wann? . . . 144
5.4.5 Verteilungssicht: Zusammenhang zur technischen Infrastruktur . . . 145
5.4.6 Querschnittliche Konzepte . . . 146
5.4.7 Entscheidungen . . . 148
5.5 Architekturdokumentation mit arc42 . . . 150
5.5.1 Aufbau von arc42 . . . 150
5.5.2 arc42 Canvas: Dokumentation kompakt . . . 152
5.6 Notationen zur Modellierung: UML, C4 und andere . . . 154
5.6.1 UML Kurzeinführung . . . 155
5.6.2 C4 Kurzeinführung . . . 159
5.6.3 Weitere Notationen für Softwarearchitektur . . . 164
5.7 Werkzeuge zur Dokumentation . . . 165
5.8 Weiterführende Literatur . . . 168

6 Analyse und Bewertung von Softwarearchitekturen . . . 169
6.1 Qualitative Architekturbewertung . . . 172

6.2 Quantitative Bewertung durch Metriken 179
6.3 Werkzeuge zur Bewertung 181

7 Technische und querschnittliche Konzepte 183
7.1 Persistenz 186
7.1.1 Motivation 186
7.1.2 Einflussfaktoren und Entscheidungskriterien 189
7.1.2.1 Art der zu speichernden Daten 190
7.1.2.2 Konsistenz und Verfügbarkeit (ACID, BASE oder CAP) 191
7.1.2.3 Zugriff und Navigation 192
7.1.2.4 Deployment und Betrieb 193
7.1.3 Lösungsmuster 193
7.1.3.1 Persistenzschicht 193
7.1.3.2 DAO: Eine Miniatur-Persistenzschicht 196
7.1.4 Bekannte Risiken und Probleme 197
7.1.5 Weitere Themen zu Persistenz 198
7.1.6 Data Contracts: Daten als Schnittstelle 202
7.1.7 Zusammenhang zu anderen Themen 204
7.1.8 Praktische Vertiefung 206
7.1.9 Weiterführende Literatur 207
7.2 Geschäftsregeln 207
7.2.1 Motivation 207
7.2.2 Funktionsweise von Regelmaschinen 210
7.2.3 Kriterien pro & kontra Regelmaschinen 212
7.2.4 Mögliche Probleme 213
7.2.5 Weiterführende Literatur 214
7.3 Integration 215
7.3.1 Motivation 215
7.3.2 Typische Probleme 217
7.3.3 Lösungskonzepte 218
7.3.4 Entwurfsmuster zur Integration 222
7.3.5 Zusammenhang mit anderen Themen 222
7.3.6 Weiterführende Literatur 223
7.4 Verteilung 224
7.4.1 Motivation 224
7.4.2 Typische Probleme 224
7.4.3 Lösungskonzept 224
7.4.4 Konsequenzen und Risiken 225
7.4.5 Zusammenhang mit anderen Themen 226
7.4.6 Weiterführende Literatur 226
7.5 Kommunikation 226
7.5.1 Motivation 227
7.5.2 Entscheidungsalternativen 227

7.5.3 Grundbegriffe der Kommunikation 227
7.5.4 Weiterführende Literatur 231
7.6 Grafische Oberflächen (GUI) 231
7.6.1 Motivation 231
7.6.2 Einflussfaktoren und Entscheidungskriterien 231
7.6.3 GUI-relevante Architekturmuster 234
7.6.4 Struktur und Ergonomie von Benutzeroberflächen 234
7.6.5 Bekannte Risiken und Probleme 236
7.6.6 Zusammenhang zu anderen Themen 238
7.7 Sicherheit 238
7.7.1 Motivation – Was ist IT-Sicherheit? 238
7.7.2 Sicherheitsziele 239
7.7.3 Lösungskonzepte 241
7.7.4 Security Engineering mit Patterns 248
7.7.5 Weiterführende Literatur 249
7.8 Protokollierung 249
7.8.1 Typische Probleme 250
7.8.2 Lösungskonzept 251
7.8.3 Zusammenhang mit anderen Themen 251
7.9 Ausnahme- und Fehlerbehandlung 252
7.9.1 Motivation 252
7.9.2 Fehlerkategorien schaffen Klarheit 254
7.9.3 Muster zur Fehlerbehandlung 256
7.9.4 Mögliche Probleme 257
7.9.5 Zusammenhang mit anderen Themen 258
7.9.6 Weiterführende Literatur 259
7.10 Skalierbarkeit 259
7.10.1 Skalierungsstrategien 259
7.10.2 Elastizität 260
7.10.3 Scale-Up-Strategie 260
7.10.4 Vertikale Scale-Out-Strategie 260
7.10.5 Horizontale Scale-Out-Strategie 261
7.10.6 Der Strategiemix 261
7.10.7 Allgemeine Daumenregeln 262
7.10.8 CPU-Power 262
7.10.9 GPU-Power 262
7.10.10 RAIDs, SANs und andere Speichersysteme 263
7.10.11 Bussysteme für die Speicheranbindung 263
7.10.12 Geringere Bandbreite im Netz 264
7.11 Container und die Cloud 264
7.11.1 Was bedeutet „Cloud“? 265
7.11.2 Virtuelle Maschinen (VMs) und Container 267
7.11.3 Von Monolithen in die Cloud 269

7.11.4 Was Sie noch über die Cloud wissen sollten.... 272
7.11.5 Weiterführende Literatur 274
7.12 Weitere spannende Themen. 274

8 Systematische Verbesserung und Evolution 277
8.1 Wege in den Abgrund. 279
8.2 Systematisch verbessern 280
8.3 Bewährte Praktiken und Muster 284
8.4 Analyse: Probleme identifizieren 285
8.5 Evaluate: Probleme und Maßnahmen bewerten 287
8.6 Improve: Verbesserungsmaßnahmen planen und durchführen 288
8.6.1 Maxime für Verbesserungsprojekte 288
8.6.2 Kategorien von Verbesserungsmaßnahmen 288
8.7 Crosscutting: phasenübergreifende Praktiken 292
8.8 Mehr zu aim[42] 293
8.9 Weiterführende Literatur 293

9 Beispiele von Softwarearchitekturen 295
9.1 Beispiel: Datenmigration im Finanzwesen 296
9.2 Beispiel: Kampagnenmanagement im CRM 313

10 iSAQB Curriculum 339
10.1 Standardisierte Lehrpläne für Softwarearchitektur 340
10.1.1 Grundlagenausbildung und Zertifizierung Foundation-Level 340
10.1.2 Fortgeschrittene Aus- und Weiterbildung (Advanced-Level) 341
10.2 iSAQB-Foundation-Level-Lehrplan. 341
10.2.1 Können, Wissen und Verstehen 342
10.2.2 Voraussetzungen und Abgrenzungen. 342
10.2.3 Struktur des iSAQB-Foundation-Lehrplans 343
10.2.4 Zertifizierung gemäß iSAQB 343
10.3 Beispielfragen zur Foundation-Level-Prüfung 344

Literatur 349

Stichwortverzeichnis 353

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.

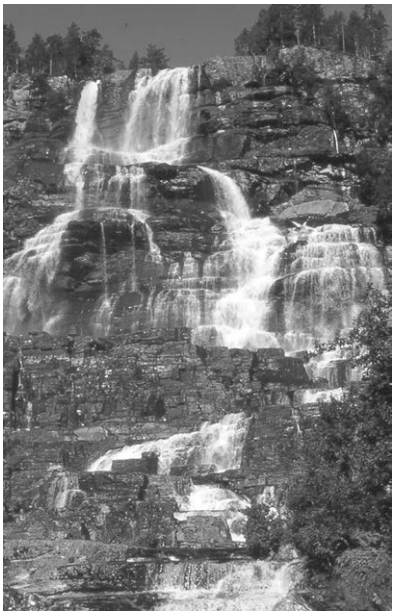

Bild 1.1 Foto von Wolfgang Korn

„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

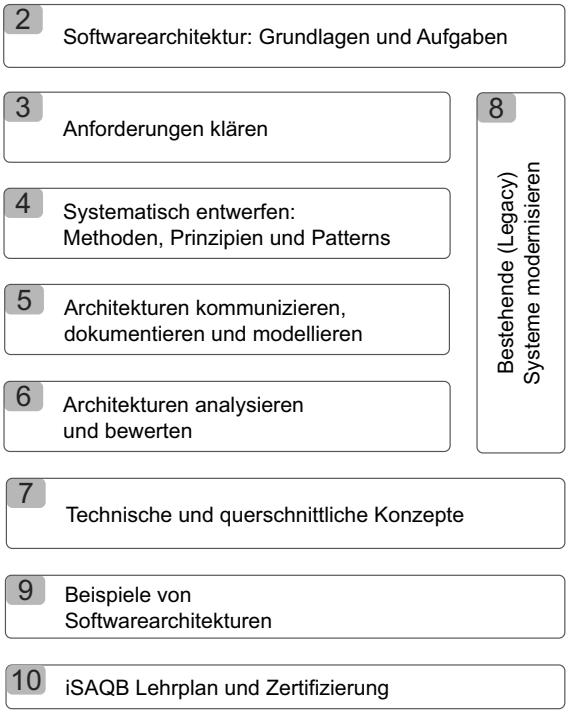

Bild 1.4
Wegweiser

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!

Stichwortverzeichnis

Symbole
12-Factor-App 274

A
Ablaufumgebung 145
Adapter 119
ADR (Architecture Decision Record) 149
Agilität 6
aim42 280
Aktivitätsdiagramm 158
Anforderungen
– an Schnittstellen 86
– klären 35
Annahmen explizit machen 287
Anpassbarkeit 77
arc42 150, 151, 152
– Aufbau 150
– Beispiele 295
– Canvas 152
ArchiMate 164
Architecture Decision Record (ADR) 149
Architecture Tradeoff Analysis Method 172
Architekt im Team 28
Architektur
– Aufgaben 22
– Bewertung 169
– degeneriert 277
– Entscheidungen 19
– Grundlagen 13
– iterativ 32
– kommunizieren 125
– mit arc42 150, 151, 152
– relevante Fragen 21
Architekturagent 29
Architekturbewertung 169
– als Teil der Architekturentwicklung 170
– ATAM 172
– Auswirkung von 178
– Vorgehen 172
Architekturdokumentation 125
– mit arc42 150
Architekturentscheidungen 19
Architektursicht 80
ATAM 172
– Qualitätsbaum 175
Aufgaben
– analysieren + bewerten 25
– Anforderungen klären 23
– dokumentieren 24
– entscheiden 24
– kommunizieren 24
– Umsetzung begleiten 25
– von Softwarearchitekt:innen 22

B
Batch-Pattern 104
Bausteinsicht 140
Baustein von Architekturen 140
Beispiel
– Kampagnenmanagement 313
– Migrationssystem Finanzwesen 296
Beispiele
– Architekturen 295
Benutzeroberfäche (GUI) 231
Beobachter 122
Bewertung
– ATAM 172
– qualitativ 179
– quantitativ 179
– Soll-Ist-Vergleich 170
– von Architekturen 169
– von Artefakten 170
– von Prozessen 170
Beziehungen, Abhängigkeiten 15
Blackboard 106
Blackbox 141
Bottom-up 79
Böxli 15

Broadcast 229
Broker 109

C
C4 159
– Code Diagram 163
– Component Diagram 160
– Container Diagram 160
– System Context 160
Chaos 4
Clean-Architecture 91
Client-Server 94
Cloud 264
Cloud-Native Computing Foundation 274
Cloud-Provider 265
Cluster
– Kubernetes 269
– von Fehlern 66
Container 267
– Orchestrierung 269
Continuous Delivery 95
Conway 32
CQRS 107
Curriculum 339
cyclomatic complexity 179

D
Definitionen, Architektur 14
Denial-of-Service 240
Dependency Inversion Principle 57
Deployment-Diagramm 158
Developer Experience 266
DIP 57
Docker 268
Dockerfile 268
Dokumentation 125
– Anforderungen an 128
– Bestandteile 138
– effektive 129
– Grundprinzipien 130
– kompakt (mit Canvas) 152
– nach arc42 150, 151, 152
– Werkzeuge 165
Domain-Driven Design 67
Domain Storytelling 38
Domänenmodell 67

E
Einfachheit 48
Einflussfaktoren 45
– organisatorische 45
– technische 46
Elastizität 260
Entscheidung 148
– unter Unsicherheit 20
Entwurf 47
– Grundlagen 48
– objektorientierter 57
– Prinzipien 51
Entwurfsmethoden 66
Entwurfsmuster
– Adapter 119
– Beobachter 122
– Fassade 121
– Observer 122
– Proxy 120
– Stellvertreter 120
Entwurfsprinzip, Schnittstellen abtrennen 59
Evaluierung von Architekturen 169
Event Storming 38
Event Systems 111
Evolution 277
Exceptions 252
explizit 49

F
Fachdomäne 38, 67
Faktoren
– organisatorische 45
– technische 46
Fassade 121
Fehlerbehandlung 252
Fehler, Cluster 66
Fehlerkategorien 254
Flexibilität 77
Foundation-Level, iSAQB 341

G
Generalisierung 63
Geschäftsziele bei Architekturbewertung 174
grafische Oberflächen 231
GraphQL 143
gRPC 143
Grundlagen
– des Entwurfs 48
– Softwarearchitektur 13
Grundprinzipien von Dokumentation 130
GUI 231
GUI-Idiome 233
GUI-Plattformen 233

H
Heuristik
– Generalisieren 63
– Spezialisieren 64
– Trial-and-Error 63
Hexagonal Architecture 91
Hyperscaler 265
Hypervisor 267

I
IaaS 265
Idiome für GUI 233
implizit 49
Improvement Backlog 283
Information Hiding 55
Infrastructure as a Service 265
Infrastruktursicht 145
Interface Segregation Principle 57
iSAQB 339
iSAQB-Lehrplan 339
ISP 57

K
k8 s 269
Kategorien von Systemen 37
KDA *siehe* Kommt-Drauf-An
Keep It Small and Simple 48
Klassendiagramm 156
Kommt-Drauf-An (KDA) 20
Komplexität 63
Komponenten 15
Komponentendiagramm 156
Konsistenz 55
– in Modellen 166
Kontext 16, 44
Kontextabgrenzung 16, 138
Kontextsicht 16
Konzepte 17, 183
– dokumentieren 146
– technische 183
Kopplung 51, 53
Kryptografie 242
Kubernetes 268

L
Latenz 98
Laufzeitsicht 144
Layer 88
Liskov Substitution Principle 57
LSP 57

M
Mengengerüst 146
Message-Queue 112, 228
Metriken 179
– für Quellcode 179
– Software 179
Microservices 94
Mindmaps als Hilfsmittel für Qualitätsbäume 175
Modelle, fachlich 67, 68, 69
Modellierung
– Klassendiagramm 156
– Komponentendiagramm 156
– Paketdiagramm 156
Model-View-Controller 114
Monarchie 27
Murphys Gesetz 50

N
NoSQL 188

O
objektorientierter Entwurf 57
Observer 122
OCP 57
Offen-Geschlossen-Prinzip 57
OpenAPI 143
Open-Closed Principle 57
Orchestrierung, Container 269
organisatorische Faktoren 45

P
PaaS 265
Paketdiagramm 156
Peer-to-Peer 110
Performance 75
Pipes und Filter 102
PlantUML 166
Platform as a Service 265
Ports-und-Adapter 91
Presentation Model 115
Prinzipien 17
– des Entwurfs 51
– SOLID 57
Protokollierung 249
Proxy 120

Q
Qualität 39
– innere 280

Qualitätsanforderungen 39
– als Bewertungsziel 172
Qualitätsbaum
– ATAM 175
– Szenarien konkretisieren 176
Qualitätsszenarien 74
quantitative Bewertung 179

R
Randbedingungen 45
Repository 105
Requirements Engineering 38
REST 117
ripple effect 277
Rolle von Softwarearchitekt:innen 26
Root-Cause-Analyse 285

S
SaaS 265
Scale-Up 260
Schätzen in Intervallen 287
Schichten 88
Schnittstellen 44, 143
– Anforderungen 86
– spezifische 59
Secure Socket Layer 247
Self-Contained System 96
Sequenzdiagramm 158
Sichten 80
– in der Softwarearchitektur 81
– -Kontextabgrenzung 138
– -Laufzeit 144
Single Responsibility Principle 57
Skalierungsstrategien 259
S/MIME 247
Softwarearchitektur
– Bewertung 169
– Definition 14
– Grundlagen 13
– Sichten 80, 81
– und Qualitat 39
Software as a Service 265
SOLID 57
Soll-Ist-Vergleich 170
Spezialisierung 64
spezifische Schnittstellen 59
SRP 57
Staging 146
Stakeholder 36
– bei Architekturbewertung 173
– maßgebliche 173
Stellvertreter 120
Struktur 16
– innere Ordnung 16
SysML 165
System 14
– Kategorien 37
Szenarien zur Bewertung 176

T
Tätigkeiten von Softwarearchitekt:innen 22
Team-Architektur 31
Teamorganisation 94
technische Faktoren 46
time-to-market 278
Top-down 79
Tracing 249
Transport Layer Security 247
Trial-and-Error 63

U
ubiquitous language 38, 67
Umgebung 16
UML 155
Umsetzung begleiten 25

V
Verallgemeinerungen 63
Verbesserung 277
– improvement backlog 283
– iteratives Vorgehen 282
– systematische 280
Verfügbarkeit 78
Verschlüsselung 241
Verteilung 224
Verteilungssicht 145
Vertraulichkeit als Sicherheitsziel 239
Virtualisierung 267
Virtual Private Networks 247
Virtuelle Maschinen 267
Vogelperspektive *siehe* Kontextabgrenzung
Vorgehen zur Architekturbewertung 172

W
Walkthrough von Szenarien 177
Werkzeuge zur Dokumentation 165
Whitebox 141

Z
Zertifikate 246