

HANSER



Leseprobe

zu

„Effektive Softwarearchitekturen“

von Gernot Starke

Print-ISBN: 978-3-446-46376-9

E-Book-ISBN: 978-3-446-46589-3

E-Pub-ISBN: 978-3-446-46690-6

Weitere Informationen und Bestellungen unter
<http://www.hanser-fachbuch.de/978-3-446-46376-9>

sowie im Buchhandel

© Carl Hanser Verlag, München

Inhalt

Vorwort	XIII
Vorwort zur neunten Auflage	XIV
1 Einleitung	1
1.1 Softwarearchitekt(inn)en	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	12
1.6 Weiterführende Literatur	12
1.7 Danksagung	13
2 Architektur und Architekten	15
2.1 Was ist Softwarearchitektur?	16
2.2 Die Aufgaben von Softwarearchitekten	21
2.3 Wie entstehen Architekturen?	26
2.4 In welchem Kontext steht Architektur?	29
2.5 Weiterführende Literatur	32
3 Vorgehen bei der Architekturentwicklung	33
3.1 Informationen sammeln	37
3.2 Anforderungen klären	38
3.2.1 Was ist die Kernaufgabe des Systems?	38
3.2.2 Welche Kategorie von System?	39
3.2.3 Wesentliche Qualitätsanforderungen ermitteln	39
3.2.4 Relevante Stakeholder ermitteln	44
3.2.5 Fachlichen und technischen Kontext ermitteln	45
3.3 Einflussfaktoren und Randbedingungen ermitteln	47
3.4 Entwerfen und kommunizieren	53
3.5 Umsetzung begleiten	54
3.6 Lösungsstrategien entwickeln	55
3.7 Weiterführende Literatur	57

4	Entwurf: Grundlagen, Methoden und Muster	59
4.1	Grundlagen	61
4.1.1	Grundsätze des Entwurfs (Maxime)	62
4.1.2	Prinzipien	65
4.1.3	SOLID-Prinzipien des objektorientierten Entwurfs	71
4.1.3.1	Offen-Geschlossen-Prinzip	71
4.1.3.2	Liskov-Substitutionsprinzip (LSP)	73
4.1.3.3	Interface Segregation Principle (ISP)	74
4.1.3.4	Dependency Inversion Principle (DIP)	76
4.2	Heuristiken	79
4.3	Entwurfsmethoden	84
4.3.1	Domain-Driven Design (Entwurf nach Fachlichkeit)	84
4.3.2	Quality-Driven Software Architecture	89
4.3.3	Top-down und Bottom-up	97
4.4	Schnittstellen entwerfen	98
4.4.1	Anforderungen an Schnittstellen	99
4.4.2	Worauf müssen Sie achten?	100
4.4.3	Tipps zum Entwurf von Schnittstellen	101
4.5	Architekturstile und -muster	102
4.5.1	Datenflussarchitekturstil	103
4.5.1.1	Architekturstil Batch-Sequentiell	103
4.5.1.2	Architekturstil Pipes und Filter	104
4.5.2	Datenzentrierter Architekturstil	107
4.5.2.1	Repository	107
4.5.2.2	Blackboard	108
4.5.3	Hierarchische Architekturstile	109
4.5.3.1	Master-Slave	109
4.5.3.2	Schichten (Layer)	110
4.5.3.3	Architekturstil Ports-und-Adapter	113
4.5.4	Architekturstile verteilter Systeme	116
4.5.4.1	Client-Server	117
4.5.4.2	Command Query Responsibility Segregation	117
4.5.4.3	Broker	119
4.5.4.4	Peer-to-Peer	120
4.5.5	Ereignisbasierte Systeme – Event Systems	121
4.5.5.1	Ungepufferte Event-Kommunikation	122
4.5.5.2	Message- oder Event-Queue-Architekturen	122
4.5.5.3	Message-Service-Architekturen	123
4.5.6	Interaktionsorientierte Systeme	124
4.5.6.1	Model-View-Controller	124
4.5.6.2	Presentation Model	125
4.5.7	Weitere Architekturstile und -muster	128
4.6	Entwurfsmuster	130
4.6.1	Entwurf mit Mustern	130

4.6.2	Adapter	131
4.6.3	Beobachter (Observer)	132
4.6.4	Dekorierer (Decorator)	133
4.6.5	Stellvertreter (Proxy)	134
4.6.6	Fassade	135
4.6.7	Zustand (State)	136
4.7	Weiterführende Literatur	137
5	Kommunikation und Dokumentation von Architekturen	139
5.1	Architekten müssen kommunizieren und dokumentieren	140
5.2	Effektive Architekturdokumentation	142
5.2.1	Anforderungen an Architekturdokumentation	142
5.2.2	Regeln für gute Architekturdokumentation	144
5.3	Typische Architekturdokumente	147
5.3.1	Zentrale Architekturbeschreibung	148
5.3.2	Architekturüberblick	151
5.3.3	Dokumentationsübersicht	151
5.3.4	Übersichtspräsentation der Architektur	151
5.3.5	Architekturtapete	152
5.4	Sichten	152
5.4.1	Sichten in der Softwarearchitektur	153
5.4.2	Vier Arten von Sichten	155
5.4.3	Entwurf der Sichten	157
5.5	Kontextabgrenzung	159
5.5.1	Elemente der Kontextabgrenzung	159
5.5.2	Notation der Kontextabgrenzung	160
5.5.3	Entwurf der Kontextabgrenzung	160
5.6	Bausteinsicht	161
5.6.1	Elemente der Bausteinsicht	165
5.6.2	Notation der Bausteinsicht	166
5.6.3	Entwurf der Bausteinsicht	167
5.7	Laufzeitsicht	168
5.7.1	Elemente der Laufzeitsicht	169
5.7.2	Notation der Laufzeitsicht	170
5.7.3	Entwurf der Laufzeitsicht	171
5.8	Verteilungssicht	171
5.8.1	Elemente der Verteilungssicht	172
5.8.2	Notation der Verteilungssicht	172
5.8.3	Entwurf der Verteilungssicht	173
5.9	Dokumentation von Schnittstellen	174
5.10	Dokumentation technischer Konzepte	177
5.11	Werkzeuge zur Dokumentation	177
5.12	TOGAF zur Architekturdokumentation	179
5.13	Weiterführende Literatur	181

6	Modellierung für Softwarearchitekten	183
6.1	Modelle als Arbeitsmittel	183
6.1.1	Grafische oder textuelle Modellierung	185
6.2	UML 2 für Softwarearchitekten	186
6.2.1	Die Diagrammarten der UML 2	187
6.2.2	Die Bausteine von Architekturen	189
6.2.3	Schnittstellen	190
6.2.4	Die Bausteinsicht	191
6.2.5	Die Verteilungssicht	193
6.2.6	Die Laufzeitsicht	195
6.2.7	Darum UML	198
6.2.8	Darum nicht UML	199
6.3	Tipps zur Modellierung	199
6.4	Weiterführende Literatur	200
7	Technische Konzepte und typische Architektur Aspekte	201
7.1	Persistenz	205
7.1.1	Motivation	205
7.1.2	Einflussfaktoren und Entscheidungskriterien	208
7.1.2.1	Art der zu speichernden Daten	209
7.1.2.2	Konsistenz und Verfügbarkeit (ACID, BASE oder CAP)	210
7.1.2.3	Zugriff und Navigation	212
7.1.2.4	Deployment und Betrieb	212
7.1.3	Lösungsmuster	213
7.1.3.1	Persistenzschicht	213
7.1.3.2	DAO: Eine Miniatur-Persistenzschicht	217
7.1.4	Bekannte Risiken und Probleme	218
7.1.5	Weitere Themen zu Persistenz	219
7.1.6	Zusammenhang zu anderen Aspekten	223
7.1.7	Praktische Vertiefung	224
7.1.8	Weiterführende Literatur	225
7.2	Geschäftsregeln	226
7.2.1	Motivation	226
7.2.2	Funktionsweise von Regelmaschinen	229
7.2.3	Kriterien pro & kontra Regelmaschinen	231
7.2.4	Mögliche Probleme	232
7.2.5	Weiterführende Literatur	233
7.3	Integration	233
7.3.1	Motivation	233
7.3.2	Typische Probleme	235
7.3.3	Lösungskonzepte	236
7.3.4	Entwurfsmuster zur Integration	240
7.3.5	Zusammenhang mit anderen Aspekten	241
7.3.6	Weiterführende Literatur	242

7.4	Verteilung	242
7.4.1	Motivation	242
7.4.2	Typische Probleme	243
7.4.3	Lösungskonzept	243
7.4.4	Konsequenzen und Risiken	245
7.4.5	Zusammenhang mit anderen Aspekten	245
7.4.6	Weiterführende Literatur	245
7.5	Kommunikation	246
7.5.1	Motivation	246
7.5.2	Entscheidungsalternativen	246
7.5.3	Grundbegriffe der Kommunikation	246
7.5.4	Weiterführende Literatur	250
7.6	Grafische Oberflächen (GUI)	251
7.6.1	Motivation	251
7.6.2	Einflussfaktoren und Entscheidungskriterien	251
7.6.3	GUI-relevante Architekturmuster	253
7.6.4	Struktur und Ergonomie von Benutzeroberflächen	254
7.6.5	Bekannte Risiken und Probleme	255
7.6.6	Zusammenhang zu anderen Aspekten	257
7.7	Geschäftsprozess-Management: Ablaufsteuerung im Großen	258
7.7.1	Workflow-Sprachen	260
7.7.2	Vorhersagbarkeit	263
7.7.3	Zweck der Ablaufsteuerung	264
7.7.4	Lösungsansätze	266
7.7.5	Integration von Workflow-Systemen	269
7.7.6	Mächtigkeit von WfMS	270
7.7.7	Weiterführende Literatur	270
7.8	Sicherheit	271
7.8.1	Motivation – Was ist IT-Sicherheit?	271
7.8.2	Sicherheitsziele	272
7.8.3	Lösungskonzepte	274
7.8.4	Security Engineering mit Patterns	281
7.8.5	Weiterführende Literatur	282
7.9	Protokollierung	283
7.9.1	Typische Probleme	283
7.9.2	Lösungskonzept	284
7.9.3	Zusammenhang mit anderen Aspekten	285
7.9.4	Weiterführende Literatur	285
7.10	Ausnahme- und Fehlerbehandlung	286
7.10.1	Motivation	286
7.10.2	Fehlerkategorien schaffen Klarheit	288
7.10.3	Muster zur Fehlerbehandlung	290
7.10.4	Mögliche Probleme	291
7.10.5	Zusammenhang mit anderen Aspekten	292

7.10.6	Weiterführende Literatur	293
7.11	Skalierbarkeit	293
7.11.1	Was bedeutet Skalierbarkeit?	293
7.11.2	Skalierungsstrategien	294
7.11.3	Elastizität	294
7.11.4	Scale-Up-Strategie	294
7.11.5	Vertikale Scale-Out-Strategie	295
7.11.6	Horizontale Scale-Out-Strategie	295
7.11.7	Der Strategiemix	295
7.11.8	Allgemeine Daumenregeln	296
7.11.9	CPU-Power	297
7.11.10	GPU-Power	297
7.11.11	RAIDs, SANs und andere Speichersysteme	297
7.11.12	Bussysteme für die Speicheranbindung	298
7.11.13	Geringere Bandbreite im Netz	298
7.12	Blockchain und dezentralisierte Architektur	299
7.12.1	Definition und Abgrenzung	299
7.12.2	Technische Grundlagen von Blockchains	301
7.12.3	Smart Contracts	302
7.12.4	Blockchains in nichtöffentlichen Kontexten	302
7.12.5	Architekturabwägungen	304
7.12.6	Architekturmuster	305
8	Analyse und Bewertung von Softwarearchitekturen	307
8.1	Qualitative Architekturbewertung	310
8.2	Quantitative Bewertung durch Metriken	317
8.3	Werkzeuge zur Bewertung	319
8.4	Weiterführende Literatur	320
9	Systematische Verbesserung und Evolution	321
9.1	Wege in den Abgrund	323
9.2	Systematisch verbessern	324
9.3	Bewährte Praktiken und Muster	327
9.4	Analyse: Probleme identifizieren	329
9.5	Evaluate: Probleme und Maßnahmen bewerten	331
9.6	Improve: Verbesserungsmaßnahmen planen und durchführen	332
9.6.1	Maxime für Verbesserungsprojekte	332
9.6.2	Kategorien von Verbesserungsmaßnahmen	332
9.7	Crosscutting: phasenübergreifende Praktiken	336
9.8	Mehr zu AIM ⁴²	337
9.9	Weiterführende Literatur	337

10	Microservices	339
10.1	Was sind Microservices?	340
10.2	Warum Microservices?	340
10.3	Eigenschaften von Microservices	341
10.4	Microservices und die Organisation	343
10.5	Für welche Systeme eignen sich Microservices?	344
10.6	Herausforderungen bei Microservices	344
10.6.1	Überblick über viele Services behalten	345
10.6.2	Microservices effektiv entwickeln	345
10.6.3	Service Discovery	346
10.6.4	UI-Integration	347
10.6.5	Dezentralisierte Daten	347
10.6.6	Versionierung von Microservices	348
10.6.7	Laufzeitumgebungen und Infrastruktur verwalten	349
10.7	Beispiele für Microservices	349
10.8	Weiterführende Literatur	349
11	Enterprise-IT-Architektur	351
11.1	Wozu Architekturebenen?	352
11.2	Aufgaben von Enterprise-Architekten	353
11.2.1	Management der Infrastrukturkosten	353
11.2.2	Management des IS-Portfolios	354
11.2.3	Definition von Referenzarchitekturen	355
11.2.4	Weitere Aufgaben	357
11.3	Weiterführende Literatur	359
12	Beispiele von Softwarearchitekturen	361
12.1	Beispiel: Datenmigration im Finanzwesen	362
12.2	Beispiel: Kampagnenmanagement im CRM	379
13	Werkzeuge für Softwarearchitekten	409
13.1	Kategorien von Werkzeugen	409
13.2	Typische Auswahlkriterien	413
14	iSAQB Curriculum	415
14.1	Standardisierte Lehrpläne für Softwarearchitekten	416
14.1.1	Grundlagenausbildung und Zertifizierung Foundation-Level	416
14.1.2	Fortgeschrittene Aus- und Weiterbildung (Advanced-Level)	417
14.2	iSAQB-Foundation-Level-Lehrplan	418
14.2.1	Können, Wissen und Verstehen	418
14.2.2	Voraussetzungen und Abgrenzungen	419
14.2.3	Lernziele des iSAQB Foundation Level	420
14.2.4	Zertifizierung gemäß iSAQB	422
14.3	Beispielfragen zur Foundation Level Prüfung	423

15	Nachwort: Architektonien	429
15.1	In sechs Stationen um die (IT-)Welt	429
15.2	Ratschläge aus dem architektonischen Manifest	432
16	Literatur	437
	Stichwortverzeichnis	441

Vorwort

*Haben Sie jemals einen dummen Fehler zweimal begangen?
- Willkommen in der realen Welt.
Haben Sie diesen Fehler hundertmal hintereinander gemacht?
- Willkommen in der Software-Entwicklung.*

Tom DeMarco,
in: „Warum ist Software so teuer?“

Wenn Sie sich für Baukunst interessieren, dann erkennen Sie sicherlich die „Handschrift“ berühmter Architekten wie Frank Lloyd Wright, Le Corbusier oder Mies van der Rohe immer wieder, egal, wo auf der Welt Sie auf Bauwerke dieser Meister stoßen. Die Funktionalität des Guggenheim Museums in New York oder des Opernhauses in Sydney gepaart mit deren Schönheit und Ästhetik sind unvergessliche Eindrücke. Das erwarten wir heute auch von unseren IT-Systemen: Funktionalität gepaart mit Stil!

Seit mehr als zwanzig Jahren versuche ich, Systementwicklern die Kunst des Architektur-entwurfs nahezubringen. Die Erfahrung hat mich gelehrt, dass man jede Person, die mit gesundem Menschenverstand ausgestattet ist, zu einem guten Systemanalytiker ausbilden kann. Softwarearchitekten auszubilden, ist wesentlich schwieriger.

Früher waren viele unserer Systeme so einfach, dass einzelne Personen die Struktur leicht im Kopf behalten konnten. Heutzutage gehört mehr dazu, um die Struktur eines Systems zu beherrschen, die Auswirkungen von Technologieentscheidungen vorausszusehen und die Vielzahl von Hilfsmitteln wie Generatoren, Frameworks, Libraries und Entwicklungswerkzeuge kosteneffizient und zielführend einzusetzen.

Viele Jahre war ich davon überzeugt, dass nur Erfahrung in der Erstellung großer Systeme und selbst gemachte Fehler gute Architekten hervorbringen. Wir wussten einfach zu wenig über Wirkungen und Folgewirkungen von Designentscheidungen. In den letzten Jahren ist die Entwicklung von Architekturen mehr und mehr zur Ingenieursdisziplin herangereift.

Gernot Starke ist es gelungen, die Essenz dieser Disziplin auf den Punkt zu bringen. Die Tipps und Tricks, die er in diesem Buch zusammengetragen hat, vermitteln Ihnen eine Fülle von Praxiserfahrungen. Selbst wenn Sie zu den Veteranen der Branche gehören, werden Sie neben vielen Déjà-vu-Erlebnissen sicherlich noch die eine oder andere Perle entdecken. Wenn Sie gerade Ihre ersten Sporen als Architekt(in) verdienen, dann können Sie sich mit den Empfehlungen den einen oder anderen Holzweg ersparen.

Trotz aller Fortschritte in der IT bleiben Konstruktion und Ausgestaltung von Architekturen dauerhaft eine Domäne für kreative Gestaltungsarbeit von Menschen und Teams. Softwarearchitekt ist daher ein Beruf mit sicherer Zukunft!

Aachen, im September 2013

Peter Hruschka

■ Vorwort zur neunten Auflage

Seit der ersten Auflage 2002 sind 18 Jahre vergangen – in denen „Softwarearchitektur“ aus den Kinderschuhen zu einer volljährigen Disziplin der Softwareentwicklung reifen konnte.

Unsere Branche hat viel gelernt, Agilität und iterativ-inkrementelle Entwicklung sind endlich im Mainstream angekommen, schnelles Feedback eher Normalität denn Ausnahme, beispielsweise durch automatisiertes Testen.

Architekturaufgaben und technische Entscheidungen liegen oftmals beim Entwicklungsteam oder werden zumindest von mehreren Personen nach Rücksprache mit den Teams getroffen – und nicht mehr von Einzelpersonen „diktiert“.

Dafür sind allerdings die Anforderungen an unsere Systeme signifikant gestiegen: Benutzerzahlen von Online-Systemen gehen oft in die Millionen, Datendurchsatz oder -volumen messen wir in Tera oder Giga, nicht mehr in Kilo ... Benutzungsschnittstellen müssen hochgradig ergonomisch sein und auf multiplen Endgeräten laufen – *zero-downtime* gehört (fast) zur Selbstverständlichkeit. Embedded- und Informationssysteme müssen oftmals sicherheitskritische Entscheidungen treffen. Ach ja – Entwicklungskosten für solche komplexen IT-Systeme sollen tendenziell sinken und neue Features am besten in kürzester Zeit zur Verfügung stehen. Für solche (krassen!) Herausforderungen benötigen Entwicklungsteams fundierte Architekturkompetenz – und genau dafür habe ich dieses Buch geschrieben.

Zur neunten Auflage: Seit der achten Auflage (2017) haben viele gründliche Leserinnen und Leser einige Dutzend kleine Fehler gefunden, die in der vorliegenden Auflage (hoffentlich) behoben sind. Ich danke allen, die mich netterweise auf diese Dinge hingewiesen haben.

Die Aktualisierungen des iSAQB-CPSA-F-Lehrplans sind komplett eingeflossen, und das Kapitel über Microservices ist von den Altlasten der SOA befreit 😊.

Meinen Kollegen Dr. Lars Hupel konnte ich dazu gewinnen, einen kleinen (aber spannenden) Exkurs zum Thema „Blockchain“ beizusteuern.

Zum Buch gibt es eine (*responsive design*) Website, die ich mittels Github¹, Jekyll und Docker pflege (<http://esabuch.de>).

Möge dieses Buch helfen, bessere Software zu entwickeln!

Köln, Juni 2020

Gernot Starke

¹ Für Insider oder falls Sie Verbesserungsvorschläge haben: <https://github.com/gernotstarke/esabuch.de-site>

1

Einleitung

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

Gerhard Chroust

Bitte erlauben Sie mir, Sie mit einer etwas böartigen 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.



Foto von Wolfgang Korn

„Wir entwerfen auch komplexe technische Systeme“, erklärt ein graumeliertes 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 Badezimmer-schranks im Obergeschoss nicht spontan beantworten kann, händigt man ihr ein Formblatt aus, das ausführlich ein Change-Management-Verfahren beschreibt.

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

Pünktlich zum Ende dieser Phase erhielt die Unternehmerin einen Ordner (zweifach) mit fast 400 Seiten Beschreibung eines Hauses. Nicht ganz das von ihr Gewünschte, weil das Entwicklungsteam aus Effizienzgründen und

um Zeit zu sparen einige (der Bauherrin nur wenig zusagende) Annahmen über die Größe mancher Räume und die Farbe einiger Tapeten getroffen hatte. Man habe zwar überall groben Sand als Bodenbelag geplant, könne das aber später erweitern. Mit etwas Zement und Wasser vermischt, stünden den Hausbewohnern später alle Möglichkeiten offen. Im Rahmen der hierbei erwarteten Änderungen habe das Team vorsorglich die Treppen als Rampe ohne Stufen geplant, um Arbeitern mit 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 mit Hilfe ausgeklügelte brachialer Methoden zu einer rotgrauen zähen Paste vermischt. Man werde sich in der Wartungsphase darum kümmern, 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 ausschließlich 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 wurde, sei völlig normal, unterrichtete sie der Projektleiter. Erst jetzt, in der zweiten Phase, würde das konkrete Objekt geplant, spezifiziert, realisiert, qualitätsgesichert und konfigurationsverwaltet.



Foto von Ralf Harder

Der Auftraggeberin wurde zu diesem Zeitpunkt auch das „Direktorat EDV“ der Firma V&V vorgestellt. Nein, diese Abteilung befasste sich nicht mit Datenverarbeitung – die Abkürzung stand für „*Einhaltung Des Vorgehensmodells*“.

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 Fenster, Türen und Wänden anzubringen, verwarf die Projektleitung mit Hinweis auf Seite 354, Aktivität PL 3.42, Paragraph 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 Gummi-

bärchen 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.



„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 „Einfamilien-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 Ingenieurdisziplinen 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

1.1 Softwarearchitekt(inn)en



Liebe Softwarearchitektinnen – ich meine überall auch Sie – obwohl ich im weiteren Buch durchgängig die maskuline (weil kürzere) Form „Softwarearchitekt“ verwende. Ich meine grundsätzlich Softwarearchitektinnen und Softwarearchitekten – verzichte aber auf künstliche Konstrukte wie Softwarearchitekt*innen. Danke für Ihr Verständnis!

Softwarearchitekten allein können diese Probleme nicht lösen. Stakeholder mit klaren Zielvorstellungen, ein motiviertes Entwicklungsteam und ein effektives, flexibles und (hoffentlich) agiles Management bilden wichtige Voraussetzungen für erfolgreiche Softwareentwicklung³. Die Rolle „Softwarearchitektur“ kommt in 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. Architekten sollten als „Anwälte der Kunden“ arbeiten. Sie sollen sicherstellen, dass die Anforderungen der Kunden einerseits umsetzbar sind und andererseits auch umgesetzt werden.

Architekten als Anwälte der Kunden

Softwarearchitekten denken langfristig – auf die gesamte Lebensdauer von IT-Systemen bezogen. Sie ermöglichen kurzfristige Änderungen, sichern gleichzeitig die Langlebigkeit und Nachhaltigkeit von Software.

Langfristigkeit

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

² Quelle: The Standish Group Chaos Report. Erhältlich unter www.standishgroup.com

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

Konzeptionelle Integrität

Softwarearchitekten 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. Das Buch gibt aktiven und angehenden Softwarearchitekten praktische Ratschläge und Hilfsmittel, diese komplexe Aufgabe effektiver zu erfüllen. Es unterbreitet konkrete Vorschläge, wie Sie als Softwarearchitekt in der Praxis vorgehen sollten. 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 beim Entwurf von Architekturen liegen.

■ 1.2 Effektiv, agil und pragmatisch

Effektivität, Agilität und Pragmatismus prägen die Grundhaltung erfolgreicher Softwarearchitekten.

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 und Architekten von Softwareprojekten müssen sich durch flexible und bedarfsgerechte Vorgehensweisen darauf einstellen. Das Schlüsselwort lautet „Agilität“.

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

- Requirements Manager müssen in Projekten 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 Produktmanager 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 in flexiblen Projekten 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 im Nachhinein um die Einhaltung starrer Vorgehensmodelle.

Softwarearchitekten müssen 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: Jeder Projektleiter, Systemanalytiker, Architekt, Tester und Entwickler überprüft ständig Risiken und entscheidet in seinem 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 (www.systemsguild.com).

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.

Effektiv = pragmatisch

Projekterfolg wird grundsätzlich vom Auftraggeber beurteilt, nicht vom Architekten. Auftraggeber wollen in erster Linie ein produktives System erhalten und nicht die Einhaltung eines starren Vorgehensmodells erzwingen. Architekten müssen daher den Zweck des Systems im Auge behalten.

Pragmatisches Vorgehen bedeutet:

- auf Dogmen und starre Vorschriften zu verzichten, wo sie nicht angemessen sind,
- zielorientiert (= effektiv) Lösungen im Sinne der Kunden zu entwickeln,
- auf Perfektionismus zu verzichten. 80 % des Ertrags erreicht man mit 20 % des Aufwands.

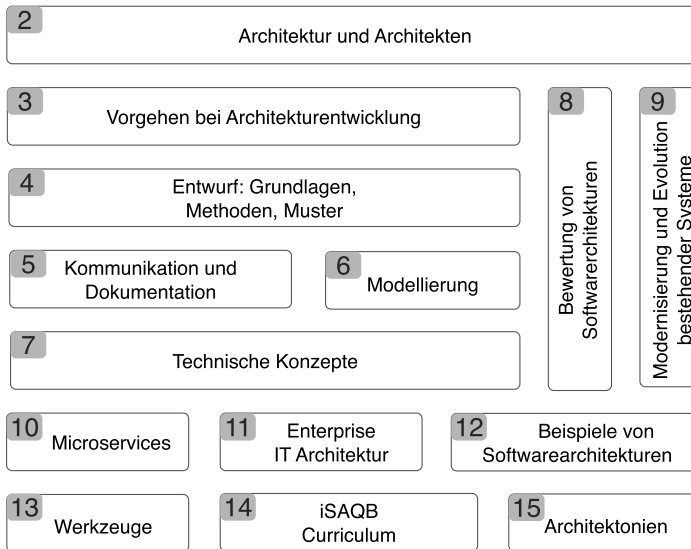
■ 1.3 Wer sollte dieses Buch lesen?

Grundsätzlich können alle Stakeholder⁴ von diesem Buch profitieren und erhalten Antworten auf zentrale Fragen.

- Softwarearchitekten
 - 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?
 - Welche Sichten auf Architekturen benötigen Sie in der Praxis?
- 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 Entwicklungsprojekten?
 - Welche Aufgaben erfüllen Softwarearchitekten?
 - Welche Bedeutung hat die Dokumentation von Architekturen („Was bedeuten all diese Symbole?“)?
 - Welche grundlegenden Lösungsansätze gibt es für Architekturen?
- Business-Analysten 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 Verlaufe 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



Kapitel 2 erläutert die Begriffe „Architektur und Architekt“. Es beantwortet die Fragen nach dem Was, Warum und Wer von Softwarearchitekturen.

Kapitel 3 legt den Grundstock für eine flexible und systematische Vorgehensweise zum Entwurf von Softwarearchitekturen. Es beschreibt die wesentlichen Aktivitäten bei der Architekturentwicklung, das Vorgehen „im Großen“. Insbesondere erfahren Sie, wie Sie aus den Faktoren, die eine Architektur beeinflussen, angemessene Lösungsstrategien ableiten können.

Kapitel 4 widmet sich dem Entwurf von Systemen. Sie lernen hier die Grundlagen des Entwurfs, geltende Prinzipien und erprobte Heuristiken. Sie finden neben Methoden für die Strukturierung Ihrer Systeme und Bausteine sowie der Gestaltung von Schnittstellen auch die wichtigsten Architektur- und Entwurfsmuster, die Sie in unterschiedlichen Bereichen der Softwareentwicklung anwenden können. Dieses Kapitel gibt Ihnen die methodischen Werkzeuge für zielgerichtete Entwurfsentscheidungen an die Hand.

arc42 Template

Kapitel 5 beschreibt, wie Sie mit praxisorientierten Sichten Ihre Softwarearchitekturen kommunizieren und dokumentieren. Jede Sicht beschreibt das System aus einer anderen Perspektive. Außerdem finden Sie hier einige Hinweise für gute Architekturdokumentation sowie das bekannte Architekturtemplate arc42.

Passend dazu fasst **Kapitel 6** die Grundlagen der Modellierung für Softwarearchitekten zusammen. Sie finden u. a. einige Basiskonstrukte der UML, abgestimmt auf die Bedürfnisse von Softwarearchitekten.

(technische) Konzepte

Kapitel 7 enthält einen Katalog häufig benötigter technischer Konzepte. Hierzu zählen Persistenz (Datenspeicherung), Integration, Verteilung, Kommunikation, Sicherheit, grafische Benutzeroberflächen, übergreifende

Stichwortverzeichnis

A

- Abhängigkeiten 59
 - zyklische 70
- Ablaufsteuerung 264
- Abschnitte, UML 186
- Abstraktionen 18, 68
 - Modellierung 183
- ACID 210
- Adapter 131
 - zur Integration 240
- Aggregate 88
- Agilität 6, 155
- aim42 324
- Aktivitätsdiagramm 187
- Änderungsszenarien 43
- Anforderungen
 - an Architekturdokumentation 142
 - an Schnittstellen 99
 - klären 38
- Anforderungsanalyse 29
- Anforderungsmanagement, Werkzeuge 409
- Anpassbarkeit 94
- Anwendungsfalldiagramm 188
- Anwendungslandschaft 351
 - Management der 354
- Anwendungsszenarien 43
- arc42 148, 149, 150
- Architecture Business Cycle 27
- Architecture Tradeoff Analysis Method 311
- Architekten
 - Aufgaben von 26
 - beraten 22
 - bewerten 24
 - entscheiden 21
 - kommunizieren 24
 - konstruieren 21
- Architektenaufgaben 53
- Architektonien 429
- Architektur
 - Bausteine 189
 - Beispiele 361
 - Bewertung 307
 - Business- 351
 - Definition 16
 - Dokumentation 139
 - Enterprise 351
 - hexagonal 113
 - Kommunikation 139
 - mit arc42 148, 149, 150
 - mit UML 2 186
 - nach Qualitätszielen 89
 - qualitativ bewerten 310
 - Unternehmens- 351
 - Werkzeuge 409
- Architektur Aspekte 201
- Architekturbeschreibung, zentrale 147
- Architekturbewertung 307
 - als Teil der Architekturentwicklung 308
 - ATAM 311
 - Auswirkung von 316
 - Vorgehen 311
- Architekturdokumentation 139, 147
 - Anforderungen 142
 - Beispiel 361
 - Grundannahmen 153
 - Regeln 144
- Architekturebenen 351
- Architekturentscheidung 16
- Architekturentwicklung, Vorgehen bei der 33
- Architekturkomitee 28
- Architekturmuster 102, 253
 - für GUI 253
- Architektursicht 152
 - Bausteinsicht 161
 - Laufzeitsicht 168
 - Verteilungssicht 171
- Architekturstil
 - Batch-Sequentiell 103

- Broker 119
- datenzentriert 107
- Event-Queue 122
- hexagonal 113
- hierarchisch 109
- interaktionsorientiert 124
- Master-Slave 109
- Messaging 122
- Microservices 340
- Model-View-Controller 124
- Peer-to-Peer 120
- Pipes und Filter 104
- Ports-und-Adapter 113
- REST 128
- Schichten/Layer 110
- Architekturtapete 152
- Architekturüberblick 151
- asynchron 247
- ATAM 311
 - Qualitätsbaum 313
- Aufgaben von Softwarearchitekten 21
- Ausbildung von Softwarearchitekten 415
- Authentifizierung 273
- Authentizität 277
- Autorisierung 273

B

- BASE 210
- Batch-Sequentiell 103
- Bausteine
 - Schnittstellen 190
 - von Architekturen 189
- Bausteinsicht 161, 162, 191
- hierarchische Verfeinerung 163
- UML 2 191
- Beispiele
 - Architekturdokumentation 361
 - von Architekturen 361
- Benutzeroberfläche (GUI) 251
- Beobachter 132
- Beschreibung von Schnittstellen 174
- Betreibbarkeit 41
- Bewertung 310
 - ATAM 311
 - qualitativ 317
 - quantitativ 317
 - Soll-Ist-Vergleich 308
 - von Architekturen 307
 - von Artefakten 308
 - von Prozessen 308

- Werkzeuge 319
- Blackboard 108
- Blackbox 162, 164
- Bottom-up 97
- Broadcast 248
- Broker 119
- Budget 49
- Buildmanagement, Werkzeuge 411
- Business-Architektur 351
- Business-IT-Alignment 358
- Business Process Management 258

C

- Caching 224
- CAP-Eigenschaften 211
- CAP-Theorem 210
- Chaos 5
- Cluster von Fehlern 83
- Codeanalyse, Werkzeuge 411
- Command-Query-Responsibility-Segregation 117
- Conway 27
- CORBA 238
- CPSA-F 416
- CQRS 117
- Curriculum 415
- cyclomatic complexity 317

D

- DAO 217
- Data Access Object 217
- Data-Binding in GUI 256
- Dateitransfer 236
- Datenbanken 206
- Datenklassen 213
- datenzentrierter Architekturstil 107
- Decorator 133
- Definition, Softwarearchitektur 16
- Dekorierer 133
- Denial-of-Service 273
- Dependency Injection 77
- Dependency Inversion Principle 71
- Design 19
 - verfaultes, Symptome 60
- Diagnostizierbarkeit 291
- Diagrammarten, UML 2 187
- Diagramm, Kontext als 46
- digitale Signatur 277
- DIN/ISO 25010 41
- DIP 71

- Dokumentation 139
 - Grundprinzipien 145
 - Konzepte 177
 - nach arc42 148, 149, 150
 - Qualitätsanforderungen 144
 - von Schnittstellen 174
 - Werkzeuge 177
- Dokumentationsübersicht 151
- Dokumente zur Beschreibung von Architekturen 147
- Domain-Driven Design 84
- Domain-Specific Languages 185
- Domänenmodell 84
- Don't Repeat Yourself 69
- DSL 185

E

- Effektiv 8
- Effizienz 8, 41
- Einfachheit 62
- Einflussfaktoren 47
 - organisatorische 48
 - technische 51
- Elastizität 294
- Enterprise-Architektur 351
- Entwurf 59
 - Grundsätze 62
 - Maxime 61
 - objektorientierter 71
 - Prinzipien 65
 - QDSA 89
- Entwurfsentscheidung 16
- Entwurfsmethoden 84
- Entwurfsmuster 130
 - Adapter 131
 - Beobachter 132
 - Decorator 133
 - Dekorierer 133
 - Fassade 135
 - Observer 132
 - Proxy 134
 - State 136
 - State (Zustand) 136
 - Stellvertreter 134
 - Zustand 136
- Entwurfsprinzip
 - abhängig nur von Abstraktionen 76
 - Dependency Injection 77
 - Liskov 73
 - Schnittstellen abtrennen 74

- Substitutionsprinzip 73
- ereignisbasierte Systeme 121
- Ergonomie 254
- Evaluierung von Architekturen 307
- Event-Queue 122
- Event Systems 121
- Evolution 321
- Exceptions 286, 287
- explizit 63

F

- Fachdomäne 84
- Fachlichkeit und Technik trennen 82
- Factories 88
- Faktoren
 - juristische 50
 - organisatorische 48
 - technische 51
- Fassade 135
 - zur Integration 240
- Fehler 83
 - Cluster 83
- Fehlerbehandlung 286, 287
- Fehlerkategorien 288
- Fingerprint 277
- Flexibilität 94
- Foundation-Level 417
- Fragen an Architekturdokumentation 142
- Funktionale Eignung 41

G

- Geheimnisprinzip 68
- Generalisierung 80
- Generierung, Werkzeuge 410
- Geschäftsprozesse, Informationsbedarf 351
- Geschäftsregeln 226
- Geschäftsziele bei Architekturbewertung 312
- grafische Oberflächen 251
- Groovy 54
- Grundprinzipien von Dokumentation 145
- Grundsätze des Entwurfs 62
- GUI 251
- GUI-Idiome 252
- GUI-Plattformen 253

H

- Hardwarearchitektur 31
- Hashfunktion 277
- Heuristik 79
 - Beachte Fehler-Cluster 83

- Fachlichkeit und Technik trennen 82
- Generalisieren 80
- Kenne die Risiken 83
- Perspektive wechseln 81
- Recherchiere 79
- Schnittstellen beachten 82
- Spezialisieren 81
- Trial-and-Error 80
- Verfeinere schrittweise 79
- hexagonale Architektur 113
- hierarchischer Architekturstil 109
- Homogenisierung 358
- HTTP 249

I

- Idiome für GUI 252
- IEEE-1471 16
- IIOIP 250
- impedance mismatch 219
- Implementierung 30
- implizit 63
- Information Hiding 68
- Informationsarchitektur 351
- Infrastruktursichten 171
- Innovation 201
- Integration 233
 - Frontend 234
- Integrität 272
 - konzeptionelle 20
 - von Daten 277
- interaktionsorientierte Systeme 124
- Interaktionsübersichtsdiagramm 188
- Interface Segregation Principle 71
- Internationalisierung 257
- iSAQB 415
- iSAQB-Foundation-Level 418
- ISP 71
- Iterationen 27
 - beim Entwurf 35
- IT-Infrastruktur 352
- IT-Sicherheit 271

J

- juristische Faktoren 50

K

- Kai-Zen 54
- Kapselung 68
- Kategorien
 - von Systemen 39

- von Werkzeugen 409
- Keep It Small and Simple 62
- KISS *siehe* Keep It Small and Simple
- Klassendiagramm 187
- Klassen, UML 2 189
- Knoten 172
 - UML 2 194
- Kohäsion 66
- Kommunikation 139, 246
 - asynchron 247
 - sync/async 247
 - von Architekturen 140
- Kommunikationsaufgabe 140
- kommunizieren 53
- Kompatibilität 42
- Komplexität 79
- Komponentendiagramm 187
- Komponenten, UML 2 189
- Konsistenz 20, 69, 210
- Kontext 45
 - als Diagramm 46
- Kontextabgrenzung 159
- Konzepte 201
 - Dokumentation 177
 - technische 201
- konzeptionelle Integrität 20
- Kopplung 65, 66
- Kryptografie 275

L

- Laufzeitsicht 168
 - UML 2 195
- Layer 110
- Lebenszyklus 20
- Legacy 233
- Lehrplan 416
- Liskov Substitution Principl 71
- Lösungsidee 38
- Lösungsstrategien 55
- LSP 71

M

- Master-Slave 109
- Maxime des Entwurfs 61
- Mediator zur Integration 240
- Message Digest 277
- Message Oriented Middleware 238
- Message Queues 247
- Messaging 122, 238
- Messgröße für Softwarearchitekturen 310

- Metriken 317
 - für Quellcode 317
 - Software 317
- Microservices 339, 343, 344
- Mindmaps als Hilfsmittel für Qualitätsbäume 313
- Modelle 183
 - fachlich 84, 85, 86
 - textbasiert 185
- Modellierung 183
 - Bausteine 189
 - grafisch 185
 - Klassendiagramm 187
 - Komponentendiagramm 187
 - Laufzeitsicht 195
 - Paketdiagramm 187
 - Schnittstellen 190
 - textuell 185
 - Tipps 199
 - Verteilung (Deployment) 193
 - Verteilungsdiagramm 187
 - Werkzeuge 410
- Model-View-Controller 124
- Modularität 68
- Module, Zerlegung in 68
- Moving Target 27
- Murphys Regel 286
- MVC 124

N

- NoSQL 207
- Notationen, grafisch/textuell 185
- Nutzen und die Ziele von Softwarearchitektur 20

O

- OAuth 278
- objektorientierter Entwurf 71
- Observer 132
- OCP 71
- Offen-Geschlossen-Prinzip 71
- Open-Closed Principle 71
- organisatorische Faktoren 48
- organisatorische Risiken 55
- organisatorische Standards 50
- OR-Mapping 219
- OSI-7-Schichten-Modell 250

P

- Paketdiagramm 187
- Pakete, UML 2 189

- Partitionstoleranz 211
- Peer-to-Peer 120
- Performance 93
- Persistenz 205
 - Einflussfaktoren 208
- Persistenzschicht 213
- Perspektive 81
- Pipes und Filter 104, 105
- Ports-und-Adapter 113
- Presentation Model 125
- Prinzipien
 - des Entwurfs 65
 - SOLID 71
- private statement *siehe* Geheimnisprinzip
- Projektplanung 30
- Protokollierung 283
- Proxy 134
- Publish-Subscribe 248

Q

- Qualität 39
- qualitative Bewertung 310
- Qualitätsanforderungen 39, 43
- Qualitätsbaum 313
 - ATAM 313
 - Szenarien konkretisieren 314
- Qualitätskriterien als Bewertungsziel 310
- Qualitätsmerkmale 41, 43, 310, 314
- Qualitätssicherung 32
- Qualitätsszenarien 91
- Qualitätsziele 35, 90
- Quality-Driven Software Architecture 89
- quantitative Bewertung 317

R

- Randbedingungen 47
- Redundanz 69
- Referenzarchitekturen 355, 356
- Regelmaschine (rule engine) 229
- Regelsysteme 229
- Remote Procedure Calls 237, 248
- Reorganisation 233
- Repositories 88
- Repository 107
- Ressourcen 49, 174
- REST 128
- REST-Architekturstil 174
- Risiken, organisatorische 55
- Risikoanalyse 31
- Risikomanagement 31

Rolle von Softwarearchitekten 15
RPC 248

S

Scale-Up 294
Schalenmodell 115
Schichten 110
Schnittstellen 45, 82, 98, 174
– Anforderungen 99
– Dokumentation von 174
– in UML beschreiben 175
– spezifische 74
– UML 2 190
– von Bausteinen 190
Secure Socket Layer 280
Security 271
Separation of Concern 67
Sequenzdiagramm 188, 196
Sicherheit 42, 271
Sicherheitsziele 272
Sichten 17, 152
– 4 Arten 155
– Arten von 155
– Baustein- 161
– für agile Architekturen 154
– in der Softwarearchitektur 153
– -Kontextabgrenzung 159
– -Laufzeit 168
– neue Arten 156
– -Verteilung 171
Signaturen, digitale 277
Single Responsibility Principle 71
Skalierbarkeit 293
Skalierung bei NoSQL 210
Skalierungsstrategien 294
S/MIME 280
Smoketest 20
SOAP 249
Softwarearchitekten 21
– Aufgaben 21
– Aufgaben von 21, 26
Softwarearchitektur 16
– Ausbildung 415
– Bewertung 307
– Definition 16
– Dokumentation und Kommunikation 140
– Iterationen 27
– Nutzen und Ziele 20
– Rolle 15
– Sichten 18, 152

– Sichten in der 153
– und Qualität 39
SOLID 71
Soll-Ist-Vergleich 308
Speichermodell 205
Speicherung 205
Spezialisierung 81
spezifische Schnittstellen 74
SRP 71
Stakeholder 44
– bei Architekturbewertung 311
– maßgebliche 311
Standardisierung 358
Standards, organisatorische 50
Starrheit 60
State 136
Stellvertreter 134
Strategien 55, 351
Strukturbruch bei Persistenz 219
Strukturen 16
Substitutionsprinzip 73
Symptome von verfaultem Design 60
synchron 247
Systeme
– ereignisbasierte 121
– interaktionsorientierte 124
– Kategorien 39
Szenarien 43
– konkretisieren Qualität 43
– zur Bewertung 314

T

TCP/IP 249
technische Faktoren 51
Test, Werkzeuge 412
Thrift 249
TOGAF 179
Top-down 97, 163
Tracing 283
Transaktionen 223
– ACID 210
Transport Layer Security 280
Trial-and-Error 80

U

Übersichtspräsentation 148
Übertragbarkeit 43
ubiquitous language 84
UML
– für Bausteinsicht 166

- für Laufzeitsicht 169
- für Schnittstellen 175
- UML 2 186
 - Aktivitäten 192
 - Diagrammarten 187
 - für Softwarearchitekten 186
 - Interaktionsübersicht 197
 - Klassen und Objekte 189
 - Knoten 194
 - Kommunikationsdiagramm 197
 - Laufzeitsicht 195
 - Schnittstellen 190
 - Verteilung 193
 - Zustände 192
- Umsetzung begleiten 54
- Unternehmensarchitektur 351

V

- Verallgemeinerungen 80
- Verantwortlichkeit trennen 67
- Verbesserung 321
- Verbesserungsprojekte 332
- verfaultes Design, Symptome 60
- Verfügbarkeit 96, 210, 273
- Verschlüsselung 274
 - symmetrisch/asymmetrisch 275
- Verschlüsselungsverfahren 275
- Verständlichkeit 18
- Verteilung 242
- Verteilungsdiagramm 187
- Verteilungssicht 171
 - UML 2 193
- Vertraulichkeit als Sicherheitsziel 272
- Virtual Private Networks 280

- Vorgehen
 - bei der Architekturentwicklung 33
 - zur Architekturbewertung 311

W

- Walkthrough von Szenarien 315
- Wartbarkeit 42
- Wasserfall 2
- Webseite 12
- Website: <http://arc42.de> 12
- Website: <https://esabuch.de> 12
- Werkzeuge 179, 409
 - Auswahlkriterien 413
 - Buildmanagement 411
 - Codemanagement 410
 - für Architekten 409
 - Kategorien 409
 - Test 412
 - zur Bewertung 319
 - zur Dokumentation 177
- Werkzeugkategorien 409
- Whitebox 162, 164
- Workflow 260

Z

- Zerbrechlichkeit 60
- Zerlegung in Module 68
- Zertifikate 279
- Zertifizierung zum CPSA-F 416
- Zugriffe auf Daten 212
- Zustand 136
- Zustandsdiagramm 188
- Zyklen in Abhängigkeiten 70
- zyklische Abhängigkeiten 70