

Mahbouba Gharbi · Arne Koschel
Andreas Rausch · Gernot Starke

Basiswissen



für Software- architekten

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



dpunkt.verlag

iSAQB



Mahbouba Gharbi ist Geschäftsführerin und Chef-Architektin bei iTech Progress GmbH und iSAQB-Vorstandsvorsitzende, ist bekenntnischer Softwarearchitektur-Fan, Autorin zahlreicher Fachartikel und häufige Sprecherin auf internationalen Konferenzen.



Prof. Dr. Arne Koschel ist Dozent an der Hochschule Hannover mit dem Schwerpunkt verteilte (Informations-)Systeme. Er hat langjährige industrielle Praxis in Entwicklung und Architektur verteilter Informationssysteme. Nebenberuflich berät und referiert er zu Themen wie SOA, Integration, Middleware, EDA und Cloud Computing. Er ist Active Board Member im iSAQB.



Prof. Dr. Andreas Rausch leitet den Lehrstuhl für Software Systems Engineering an der Technischen Universität Clausthal. Er war und ist in der industriellen Praxis als Berater und leitender Softwarearchitekt bei einer Reihe von großen verteilten Softwaresystemen tätig.



Dr. Gernot Starke, innoQ Fellow, arbeitet als Berater für methodische Softwarearchitektur, Technologiemanagement und Projektorganisation. Seit mehr als 15 Jahren gestaltet er die Architektur von Softwaresystemen unterschiedlicher Größe.

Mahbouba Gharbi · Arne Koschel · Andreas Rausch · Gernot Starke

Basiswissen für Softwarearchitekten

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

4., überarbeitete und aktualisierte Auflage



dpunkt.verlag

Mahbouba Gharbi
m.gharbi@itech-progress.com

Arne Koschel
akoschel@acm.org

Andreas Rausch
andreas.rausch@tu-clausthal.de

Gernot Starke
gs@gernotstarke.de

Lektorat: Christa Preisendanz
Copy-Editing: Ursula Zimpfer, Herrenberg

Satz: Birgit Bäuerlein

Herstellung: Stefanie Weidner

Umschlaggestaltung: Helmut Kraus, www.exclam.de

Druck und Bindung: mediaprint solutions GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek

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

ISBN:

Print 978-3-86490-781-4

PDF 978-3-96910-012-7

ePub 978-3-96910-013-4

mobi 978-3-96910-014-1

4., überarbeitete und aktualisierte Auflage 2020

Copyright © 2020 dpunkt.verlag GmbH

Wieblinger Weg 17

69123 Heidelberg

Hinweis:

Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.



Schreiben Sie uns:

Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: hallo@dpunkt.de.

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.

Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor noch Verlag können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.

Vorwort zur 4. Auflage

Softwarearchitektur bildet – neben motivierten Teams und gutem Management – einen wichtigen Erfolgsfaktor von Softwareprojekten. Sie stellt im Sinne einer systematischen Konstruktion sicher, dass Qualitätsanforderungen wie beispielsweise Erweiterbarkeit, Flexibilität, Performance oder Time-to-Market erfüllt werden können.

Softwarearchitektinnen und Softwarearchitekten bringen die Kundenwünsche in Einklang mit den technischen Möglichkeiten und Randbedingungen. Sie sorgen für eine passende Struktur und das Zusammenspiel aller Systemkomponenten. Als Teamplayer arbeiten sie eng mit Softwareentwicklerinnen und Softwareentwicklern sowie anderen Projektbeteiligten zusammen.

Unser Buch »Basiswissen für Softwarearchitekten« orientiert sich am Lehrplan zum »Certified Professional for Software Architecture – Foundation Level« (CPSA-F) des International Software Architecture Qualification Board (iSAQB). Der iSAQB e.V. legt als internationales und offenes Gremium Standards für die Ausbildung, Prüfung und Zertifizierung von Softwarearchitekten fest.

Die 4. Auflage unseres Buches bietet eine Aktualisierung auf Basis des neuen CPSA-F-Lehrplans in der Version 5.1 vom Januar 2020. Bei der Überarbeitung des iSAQB-Lehrplans wurden einige Themen auf weitere Ausbildungsstufen verschoben und sind somit nicht mehr Teil des »Foundation Level«-Lehrplans. Diese Inhalte sind zwar weiterhin in unserem Buch zu finden, sie sind jedoch als »Exkurs« hervorgehoben. Wer das Buch nur zur Prüfungsvorbereitung nutzt, der kann diese Exkurse ignorieren. Des Weiteren wurde das Glossar aktualisiert. Die Leserinnen und Leser¹ können sich auch auf neue und verbesserte Prüfungsbeispielfragen freuen, die eine gezieltere Prüfungsvorbereitung ermöglichen.

Mit der Zertifizierung zum CPSA-F weisen Softwarearchitekten einen fundierten Wissens- und Kenntnisstand für die Konstruktion kleiner und mittlerer Systeme nach. Ausgehend von einer hinreichend detailliert beschriebenen Anfor-

1. Im weiteren Verlauf des Buches verwenden wir überwiegend die männliche Form und wollen damit Frauen sowie alle anderen Geschlechter selbstverständlich nicht ausschließen bzw. ausgrenzen.

derungsspezifikation können sie eine angemessene Softwarearchitektur entwerfen und dokumentieren. CPSA-F-Absolventen besitzen damit das Rüstzeug, um problembezogene Entwurfsentscheidungen auf der Basis ihrer vorab erworbenen Praxiserfahrung zu treffen.

Das Selbststudium des vorliegenden Buches ermöglicht die Vorbereitung auf diese Zertifizierungsprüfung – praktische Erfahrung in Entwurf und Entwicklung von Softwaresystemen, das Beherrschung einer höheren Programmiersprache sowie der Grundlagen von UML vorausgesetzt. Darüber hinaus empfehlen wir als Autoren grundsätzlich den Besuch entsprechender Präsenzveranstaltungen, weil der Erfahrungsaustausch mit anderen Experten nicht durch Lektüre zu ersetzen ist.

Wir als Autoren arbeiten, lehren und forschen seit vielen Jahren im Bereich des Software & Systems Engineering sowie zur Konstruktion mittlerer und großer IT-Systeme. Wir hoffen, einen Teil unserer Erfahrungen in diesem Buch für Sie als Leser angemessen aufbereitet zu haben.

Wir wünschen Ihnen viel Spaß beim Lesen sowie viel Erfolg bei Ihrer Schulungsmaßnahme und Prüfung zum CPSA-F.

Mahbouba Gharbi, Arne Koschel, Andreas Rausch, Gernot Starke
Ludwigshafen, Hannover, Clausthal-Zellerfeld, Köln, im Juni 2020

Inhaltsübersicht

1	Einleitung	1
1.1	Softwarearchitektur als Disziplin im Software Engineering	2
1.2	iSAQB – International Software Architecture Qualification Board	4
1.3	Certified Professional for Software Architecture – Foundation und Advanced Level	5
1.4	Zielsetzung des Buches	7
1.5	Voraussetzungen	8
1.6	Leitfaden für den Leser	9
1.7	Zielpublikum	10
1.8	Danksagungen	10
2	Grundlagen von Softwarearchitekturen	11
2.1	Einbettung in den iSAQB-Lehrplan	12
2.2	Softwareintensive Systeme und Softwarearchitekturen	13
2.3	Grundlegende Konzepte von Softwarearchitekturen	20
2.4	Der Softwarearchitekturentwurf aus der Vogelperspektive	38
2.5	Lernkontrolle	48
3	Entwurf von Softwarearchitekturen	51
3.1	Einbettung in den iSAQB-Lehrplan	52
3.2	Überblick über das Vorgehen beim Architekturentwurf	52
3.3	Entwurfsprinzipien und Heuristiken	59
3.4	Architekturzentrierte Entwicklungsansätze	64
3.5	Techniken für einen guten Entwurf	72
3.6	Architekturmuster	79
3.7	Entwurfsmuster	90
3.8	Lernkontrolle	96

4	Beschreibung und Kommunikation von Softwarearchitekturen	99
4.1	Einbettung in den iSAQB-Lehrplan	99
4.2	Das CoCoME-Beispiel	100
4.3	Sichten und Schablonen	103
4.4	Technische oder querschnittliche Konzepte in Softwarearchitekturen	132
4.5	Architektur und Implementierung	135
4.6	Übliche Dokumenttypen für Softwarearchitekturen	137
4.7	Praxisregeln zur Dokumentation	140
4.8	Beispiele weiterer Architektur-Frameworks	143
4.9	Lernkontrolle	146
5	Softwarearchitekturen und Qualität	149
5.1	Einbettung in den iSAQB-Lehrplan	150
5.2	Bewertung von Softwarearchitekturen	151
5.3	EXKURS: Prototyp und technischer Durchstich	159
5.4	Architekturanalyse	161
5.5	Lernkontrolle	169
6	EXKURS: Werkzeuge für Softwarearchitekten	171
6.1	Allgemeine Hinweise zu Werkzeugen	171
6.2	Werkzeuge zum Anforderungsmanagement	172
6.3	Werkzeuge zur Modellierung	174
6.4	Werkzeuge zur Generierung	175
6.5	Werkzeuge zur statischen Codeanalyse	176
6.6	Werkzeuge zur dynamischen Analyse	177
6.7	Werkzeuge zum Build-Management	179
6.8	Werkzeuge zum Konfigurations- und Versionsmanagement	180
6.9	Werkzeuge zum Codemanagement	181
6.10	Werkzeuge zum Test	182
6.11	Werkzeuge zur Dokumentation	183

Anhang	185
A Beispielfragen	187
A.1 Auszüge aus der Prüfungsordnung	187
A.2 Beispielfragen	189
B Abkürzungsverzeichnis	193
C Glossar	195
D Literaturverzeichnis	209
Index	215

Inhaltsverzeichnis

1	Einleitung	1
1.1	Softwarearchitektur als Disziplin im Software Engineering	2
1.2	iSAQB – International Software Architecture Qualification Board	4
1.3	Certified Professional for Software Architecture – Foundation und Advanced Level	5
1.4	Zielsetzung des Buches	7
1.5	Voraussetzungen	8
1.6	Leitfaden für den Leser	9
1.7	Zielpublikum	10
1.8	Danksagungen	10
2	Grundlagen von Softwarearchitekturen	11
2.1	Einbettung in den iSAQB-Lehrplan	12
2.1.1	Lernziele	12
2.2	Softwareintensive Systeme und Softwarearchitekturen	13
2.2.1	Was ist ein softwareintensives System?	13
2.2.2	EXKURS: Ausprägungen von softwareintensiven Systemen	15
2.2.3	Bedeutung der Softwarearchitektur für ein softwareintensives System	19
2.3	Grundlegende Konzepte von Softwarearchitekturen	20
2.3.1	Was ist eine Softwarearchitektur?	21
2.3.2	Bausteine, Schnittstellen und Konfigurationen	22
2.3.3	Konzepte der Beschreibung von Softwarearchitekturen	29
2.3.4	Architekturbeschreibung und Architekturebenen	33
2.3.5	Wechselwirkungen zwischen Softwarearchitektur und Umgebung	35
2.3.6	Qualität und Nutzen der Softwarearchitektur	37

2.4	Der Softwarearchitekturentwurf aus der Vogelperspektive	38
2.4.1	Ziele und Aufgaben des Softwarearchitekturentwurfs	39
2.4.2	Der Softwarearchitekturentwurf im Überblick	41
2.4.3	Wechselspiel der Tätigkeiten und Abstraktionsstufen im Entwurf	43
2.4.4	EXKURS: Aufgaben des Softwarearchitekten und Bezug zu anderen Rollen	46
2.5	Lernkontrolle	48
3	Entwurf von Softwarearchitekturen	51
3.1	Einbettung in den iSAQB-Lehrplan	52
3.1.1	Lernziele	52
3.2	Überblick über das Vorgehen beim Architekturentwurf	52
3.3	Entwurfsprinzipien und Heuristiken	59
3.3.1	Top-down und bottom-up	59
3.3.2	Hierarchische (De-)Komposition	61
3.3.2.1	Divide et impera	61
3.3.2.2	Prinzipien bei der Zerlegung	61
3.3.2.3	So-einfach-wie-möglich-Prinzip	62
3.3.2.4	Trennung von Verantwortlichkeiten	62
3.3.3	Schmale Schnittstellen und Information Hiding	63
3.3.3.1	Information Hiding	63
3.3.3.2	Verwendung von Schnittstellen	63
3.3.4	Regelmäßiges Refactoring und Redesign	63
3.4	Architekturzentrierte Entwicklungsansätze	64
3.4.1	EXKURS: Domain Driven Design	65
3.4.1.1	Fachmodelle als Basis	65
3.4.1.2	Systematische Verwaltung der Domänenobjekte .	66
3.4.1.3	Strukturierung der Fachdomäne	66
3.4.1.4	Arten von Domänen	67
3.4.1.5	Integration von Domänen	67
3.4.2	EXKURS: MDA	68
3.4.3	Referenzarchitekturen	70
3.4.3.1	Generative Erzeugung von Systembausteinen .	70
3.4.3.2	Aspektorientierung	70
3.4.3.3	Objektorientierung	71
3.4.3.4	Prozedurale Ansätze	72

3.5	Techniken für einen guten Entwurf	72
3.5.1	Ausgangssituation und Motivation: degeneriertes Design ..	73
3.5.2	Lose Kopplung	74
3.5.3	Hohe Kohäsion	75
3.5.4	Offen-geschlossen-Prinzip	76
3.5.5	Umkehr der Abhängigkeiten	76
3.5.6	Abtrennung von Schnittstellen	77
3.5.7	Zyklische Abhängigkeiten auflösen	78
3.5.8	Liskov'sches Substitutionsprinzip	78
3.6	Architekturmuster	79
3.6.1	Adaptierbare Systeme	80
3.6.1.1	Dependency Injection	80
3.6.2	Interaktive Systeme	81
3.6.2.1	Model View Controller	81
3.6.2.2	Model View Presenter	82
3.6.2.3	Presentation Abstraction Control	83
3.6.3	Vom Chaos zur Struktur	84
3.6.3.1	Schichtenarchitektur	84
3.6.3.2	Pipes and Filters	85
3.6.3.3	Blackboard	86
3.6.4	Verteilte Systeme	86
3.6.4.1	Broker	87
3.6.4.2	EXKURS: Serviceorientierung	88
3.6.4.3	Modularisierung	89
3.6.4.4	Microservices	89
3.7	Entwurfsmuster	90
3.7.1	Adapter	90
3.7.2	Observer	91
3.7.3	Decorator	92
3.7.4	Proxy	92
3.7.5	Fassade	93
3.7.6	Brücke	94
3.7.7	State	94
3.7.8	Mediator	95
3.8	Lernkontrolle	96

4	Beschreibung und Kommunikation von Softwarearchitekturen	99
4.1	Einbettung in den iSAQB-Lehrplan	99
4.1.1	Lernziele	100
4.2	Das CoCoME-Beispiel	100
4.2.1	Anwendungsfälle im CoCoME-System	101
4.2.2	Übersicht über den strukturellen Aufbau des CoCoME-Systems	102
4.3	Sichten und Schablonen	103
4.3.1	Bewährte Sichten nach iSAQB	103
4.3.2	UML-Diagramme als Notationsmittel in Sichtenbeschreibungen	105
4.3.3	Sichtenbeschreibung – Grobaufbau und Einführungsbeispiel	108
4.3.3.1	Grobaufbau – schablonenartige Sichtenbeschreibung	108
4.3.3.2	Beispiel: Auszug aus einer Sichtenbeschreibung für eine Bausteinsicht	110
4.3.4	Kontextsicht oder Kontextabgrenzung	112
4.3.5	Bausteinsicht	116
4.3.6	Laufzeitsicht	119
4.3.7	Verteilungssicht bzw. Infrastruktursicht	124
4.3.8	Wechselwirkungen zwischen Architektursichten	128
4.3.9	Hierarchische Verfeinerung von Architektursichten	129
4.4	Technische oder querschnittliche Konzepte in Softwarearchitekturen	132
4.4.1	Technische bzw. querschnittliche Konzepte: Beispieldimensionen	133
4.4.2	Beispiel: Fehlerbehandlung	133
4.4.3	Beispiel: Sicherheit	134
4.5	Architektur und Implementierung	135
4.5.1	Beispiel: Implementierung	136
4.6	Übliche Dokumenttypen für Softwarearchitekturen	137
4.6.1	Zentrale Architekturbeschreibung	137
4.6.2	Architekturüberblick	138
4.6.3	Dokumentübersicht	138
4.6.4	Übersichtspräsentation	138
4.6.5	»Architekturtapete«	139
4.6.6	Handbuch zur Dokumentation	139

4.6.7	Technische Informationen	139
4.6.8	Dokumentation von externen Schnittstellen	139
4.6.9	Template	140
4.7	Praxisregeln zur Dokumentation	140
4.7.1	Regel 1: »Schreiben aus der Sicht des Lesers«	140
4.7.2	Regel 2: »Unnötige Wiederholung vermeiden«	141
4.7.3	Regel 3: »Mehrdeutigkeit vermeiden«	141
4.7.4	Regel 4: »Standardisierte Organisationsstruktur bzw. Schablonen«	141
4.7.5	Regel 5: »Begründen Sie wesentliche Entscheidungen schriftlich«	142
4.7.6	Regel 6: »Überprüfung auf Gebrauchstauglichkeit«	142
4.7.7	Regel 7: »Übersichtliche Diagramme«	142
4.7.8	Regel 8: »Regelmäßige Aktualisierungen«	143
4.8	Beispiele weiterer Architektur-Frameworks	143
4.8.1	4+1-Framework	144
4.8.2	RM-ODP	144
4.8.3	SAGA	146
4.9	Lernkontrolle	146
5	Softwarearchitekturen und Qualität	149
5.1	Einbettung in den iSAQB-Lehrplan	150
5.1.1	Lernziele	150
5.2	Bewertung von Softwarearchitekturen	151
5.2.1	Qualitative Bewertung	151
5.2.1.1	DIN ISO/IEC 25010	151
5.2.1.2	Qualitätsmerkmale	151
5.2.1.3	Weitere Qualitätsmerkmale	153
5.2.1.4	Auswirkungen bestimmter Qualitätsmerkmale .	154
5.2.1.5	Taktiken und Praktiken	154
5.2.2	Quantitative Bewertung	156
5.2.2.1	Überprüfung von Architekturregeln	156
5.2.2.2	Metriken	157
5.2.2.3	Zyklomatische Komplexität	158
5.3	EXKURS: Prototyp und technischer Durchstich	159
5.3.1	Technischer Durchstich	159
5.3.2	Prototyp	159
5.3.2.1	Einsatz von Softwareprototypen	159
5.3.2.2	Arten von Softwareprototypen	160

5.4	Architekturanalyse	161
5.4.1	EXKURS: ATAM-Methode	161
5.4.1.1	Vorgehen bei der Bewertung	161
5.5	Lernkontrolle	169
6	EXKURS: Werkzeuge für Softwarearchitekten	171
6.1	Allgemeine Hinweise zu Werkzeugen	171
6.1.1	Kosten von Werkzeugen	171
6.1.2	Lizenzen und Lizenzbedingungen	172
6.2	Werkzeuge zum Anforderungsmanagement	172
6.2.1	Anforderungen und Entscheidungskriterien	173
6.2.2	Herausforderungen von Werkzeugen für das Anforderungsmanagement	173
6.2.3	Beispielhafte Vertreter	173
6.3	Werkzeuge zur Modellierung	174
6.3.1	Anforderungen und Entscheidungskriterien	174
6.3.2	Herausforderungen von Werkzeugen für die Modellierung	175
6.3.3	Beispielhafte Vertreter	175
6.4	Werkzeuge zur Generierung	175
6.4.1	Anforderungen und Entscheidungskriterien	176
6.4.2	Herausforderungen von Codegeneratoren	176
6.4.3	Beispielhafte Vertreter	176
6.5	Werkzeuge zur statischen Codeanalyse	176
6.5.1	Anforderungen und Entscheidungskriterien	177
6.5.2	Herausforderungen von Werkzeugen zur statischen Codeanalyse	177
6.5.3	Beispielhafte Vertreter	177
6.6	Werkzeuge zur dynamischen Analyse	177
6.6.1	Anforderungen und Entscheidungskriterien	178
6.6.2	Herausforderungen von Werkzeugen zur dynamischen Analyse	178
6.6.3	Beispielhafte Vertreter	178
6.7	Werkzeuge zum Build-Management	179
6.7.1	Anforderungen und Entscheidungskriterien	179
6.7.2	Herausforderungen von Werkzeugen zum Build-Management	179
6.7.3	Beispielhafte Vertreter	180

6.8	Werkzeuge zum Konfigurations- und Versionsmanagement	180
6.8.1	Anforderungen und Entscheidungskriterien	180
6.8.2	Herausforderungen von Werkzeugen zum Konfigurations- und Versionsmanagement	181
6.8.3	Beispielhafte Vertreter	181
6.9	Werkzeuge zum Codemanagement	181
6.9.1	Herausforderungen von Werkzeugen zum Codemanagement	182
6.9.2	Beispielhafte Vertreter	182
6.10	Werkzeuge zum Test	182
6.10.1	Anforderungen und Entscheidungskriterien	183
6.10.2	Herausforderungen von Testwerkzeugen	183
6.10.3	Beispielhafte Vertreter	183
6.11	Werkzeuge zur Dokumentation	183
6.11.1	Anforderungen und Entscheidungskriterien	183
6.11.2	Herausforderungen von Dokumentationswerkzeugen	184
6.11.3	Beispielhafte Vertreter	184
Anhang		185
A	Beispielfragen	187
A.1	Auszüge aus der Prüfungsordnung	187
A.2	Beispielfragen	189
B	Abkürzungsverzeichnis	193
C	Glossar	195
D	Literaturverzeichnis	209
	Index	215

1 Einleitung

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

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

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

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

1.1 Softwarearchitektur als Disziplin im Software Engineering

Bereits in den 60er-Jahren wurden die Probleme mit Softwareprojekten unter dem Stichwort Softwarekrise bekannt. Vom 7. bis 11. Oktober 1968 fand im oberbayerischen Garmisch eine kleine Konferenz statt: Das Wissenschaftskomitee der NATO hatte 62 hochrangige Forscher und Praktiker von internationalem Ruf eingeladen, um unter dem Titel »Software Engineering« über die Zukunft der Softwareentwicklung nachzudenken. Heute gilt diese Konferenz als Geburtsstunde des Software Engineering [Dij72].

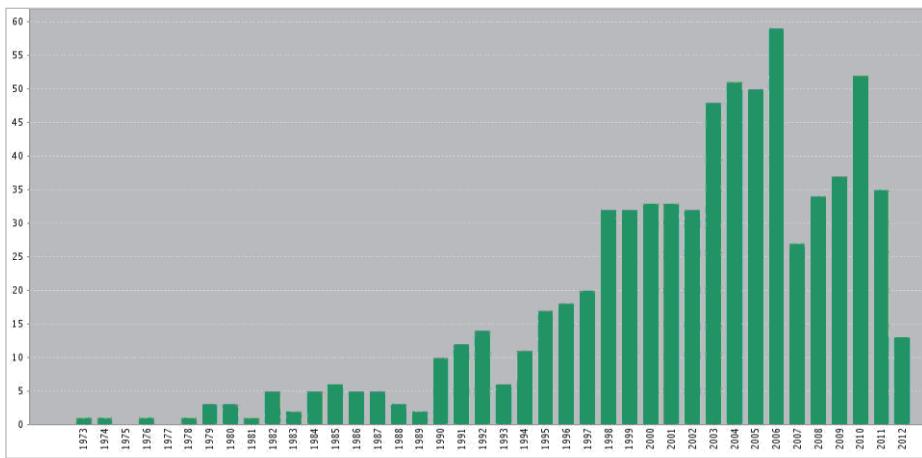


Abb. 1-1 Veröffentlichungen zu Softwarearchitektur seit 1973 [Reu12]

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

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

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



Abb. 1-2 Architektur im alten Rom

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

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

1.2 iSAQB – International Software Architecture Qualification Board

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

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

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

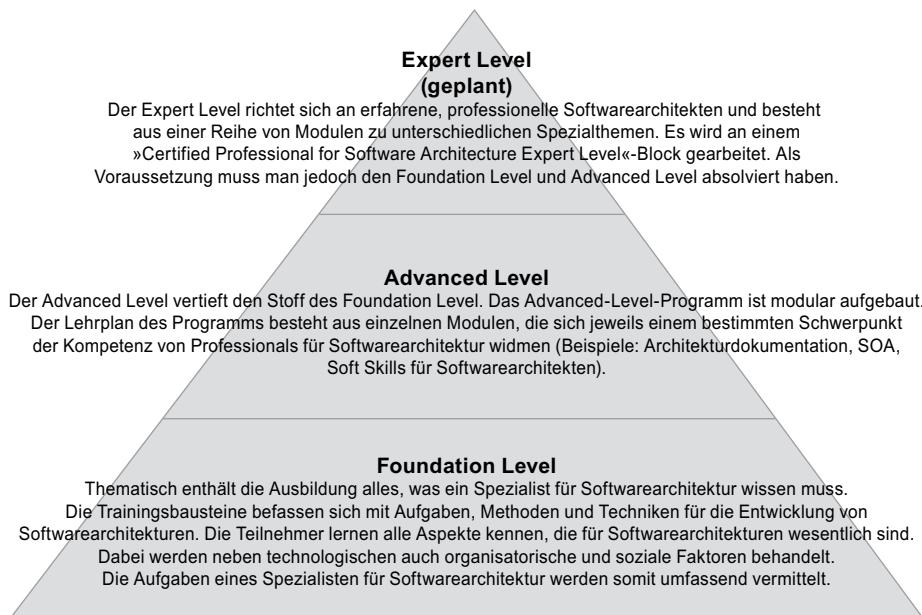


Abb. 1–3 iSAQB-Zertifizierungsstufen (www.isaqb.org)

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

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

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

1.3 Certified Professional for Software Architecture – Foundation und Advanced Level

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

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

■ **Methodische Kompetenz:**

Wissen und Fähigkeiten im Bereich des systematischen Vorgehens bei IT-Projekten, unabhängig von Technologien

■ **Technische Kompetenz:**

Wissen und Fähigkeiten im Bereich des Einsatzes von Technologien zur Lösung von Entwurfsaufgaben

■ **Kommunikative Kompetenz:**

Wissen und Fähigkeiten im Bereich der Kommunikation, Präsentation, Rhetorik und Moderation zur effektiven Wahrnehmung der Rolle im Softwareentwicklungsprozess

Voraussetzungen für den Advanced Level sind:

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

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

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

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

Im Mittelpunkt steht der Erwerb folgender Fähigkeiten:

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

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

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