Table of Contents

Preface 1

Chapter 1: The Meaning of Software Architecture 9

What is software architecture? 9

ISO/IEC/IEEE 42010 standard definition 10

What makes up a software architecture? 11

Software architecture is an abstraction 11

Software architecture is about the important stuff 11

Why is software architecture important? 12

Defining a solution to meet requirements 12

Enabling and inhibiting quality attributes 13

Giving you the ability to predict software system qualities 13

Easing communication among stakeholders 14

Managing change 14

Providing a reusable model 15

Imposing implementation constraints 15

Improving cost and effort estimates 15

Serves as training for team members 16

Software architecture is not a silver bullet 17

Who are the consumers of software architectures? 17

What is the software architect role? 18

Software architects are technical leaders 18

Software architects perform a number of duties 19

Ivory tower software architects 19

What are software architects expected to know? 20

Don't be overwhelmed 21

Is the software architect role right for you? 22

Summary 23

Chapter 2: Software Architecture in an Organization 25

Types of software architects 26

Enterprise architect 26

Solution architect 27

Application architect 27

Data architect/information architect 28

Infrastructure architect 29

Information security architect 30

Cloud architect 30

Software development methodologies 31

The Waterfall model 32

Phases of the Waterfall methodology 33
Issues with the Waterfall methodology 34
Agile software development methodologies 34
Agile values and principles 35
An iterative methodology 36
Adaptive rather than predictive 36
Daily stand-up meetings 37
Project management 38
The importance of software project estimation 38
Putting effort into the estimates 38
Being a realist (or even a pessimist) 39
Team and situational factors to consider 39
Project schedule changes 39
Getting a project back on schedule 40
Working overtime 40
Reducing scope 40
Adding resources 41
Reallocating resources 41
Identifying problem areas 42
Acting as early as possible 42
Office politics 42
Understanding your organization's goals 44
Addressing the concerns of others 44
Assisting people with their goals 44
Knowing when to compromise 45
Being aware of cultural differences 45
Software risk management 45
Risk avoidance 47
Transferring the risk to another party 47
Risk mitigation 48
Risk acceptance 48
Configuration management 48
Changing management 49
Software product lines 51
Benefits of a software product line 52
Core assets of an organization 52
Risks of product line engineering 53
Summary 53
Chapter 3: Understanding the Domain 55
Developing business acumen 55
Familiarity with general business topics 56
Understanding your organization's business 57
Domain-driven design 58
Encourages and improves communication 58
What is a ubiquitous language? 59

Entities, value objects, and aggregates 61
- Entities 61
- Value objects 61
- Aggregates and root entities 62

Separating the domain into subdomains 62
What are bounded contexts? 63

Requirements engineering 65

Types of software requirements 65
- Business requirements 66
- Functional requirements 66
- Non-functional requirements 67
- Constraints 67

The importance of requirements engineering 68
Software requirements must be measurable and testable 69
Software requirements that affect architecture 69

Requirements elicitation 70

Techniques to elicit requirements 71
- Interviews 71
- Requirements workshops 71
- Brainstorming 72
- Observation 73
- Focus groups 74
- Surveys 74
- Document analysis 75
- Prototyping 75
- Reverse engineering 76

Get access to the proper stakeholders 76

Summary 77

Chapter 4: Software Quality Attributes 79

Quality attributes 79

External or internal 80
Quality attributes and the SDLC 81
Testing quality attributes 81

Maintainability 82

Types of software maintenance 83
- Corrective maintenance 83
- Perfective maintenance 83
- Adaptive maintenance 84
- Preventive maintenance 84

Modifiability 84
Extensibility and flexibility 85
Scope of modifications 85
Designing for maintainability 85
- Reducing size 86
- Increasing cohesion 86
- Reducing coupling 86

Measuring maintainability 86

Lines of code (LOC) 87
Cyclomatic complexity 87
Depth of inheritance tree (DIT) 89

Usability 90
Allowing users to complete their tasks efficiently 90
Learnability 90
Providing useful feedback 91
Accessibility 91
Usability needs to be considered during requirements 92
Usability testing 92
Appealing visual design 92
Providing a good help system 94
Software must be useful, and not just usable 94

Availability 94
Calculating availability based on time 95
Calculating availability based on request success rate 96
Faults, errors, and failures 96
Detecting faults 97
Ping/echo reply 97
Heartbeat 97
Timestamp 97
Voting 97
Sanity test/sanity checking 98
Condition monitoring 98
Self-tests 98
Recovering from faults 98
Exception handling 98
Retry strategy 99
Varying levels of redundancy 99
Rollback 100
Graceful degradation 100
Ignoring faulty behavior 100
Preventing faults 101
Removal from service 101
Transactions 101
Increasing competence sets 101
Exception prevention 101

Portability 102
Adaptability 102
Installability 102
Replaceability 103
Internationalization and localization 103
Maintaining portability 105

Interoperability 105
Challenges with interoperability 105
Locating and exchanging information with another system 106
Interoperability standards 107

Interoperability testing 107

Testability 108

Controllability 108

Observability 109

Isolability 109

Automatability 109

Complexity of the software 110

Importance of test documentation 111

What makes a good tester? 112

Summary 113

Chapter 5: Designing Software Architectures 115

Software architecture design 116

Making design decisions 116

Software architecture design terms 117

Structure 117

Element 118

System 118

Subsystem 118

Module 118

Component 119

The importance of software architecture design 119

Making key decisions 119

Avoiding design decisions can incur technical debt 120

Communicating the architecture to others 120

Providing guidance to developers 121

Influencing non-technical parts of the project 121

Top-down versus bottom-up design approaches 121

Top-down approach 121

Advantages of the top-down approach 122

Disadvantages of the top-down approach 122

Bottom-up approach 123

Advantages of the bottom-up approach 124

Disadvantages of the bottom-up approach 124

Which approach should I use? 125

Greenfield versus brownfield software systems 127

Greenfield systems 127

Brownfield systems 128

Architectural drivers 128

Design objectives 129

Primary functional requirements 130

Quality attribute scenarios 130

Prioritizing quality attribute scenarios 131

Constraints 131

Architectural concerns 132

Leveraging design principles and existing solutions 133

Selecting a design concept 133
Software architecture patterns 134
Reference architectures 134
Benefits of reference architectures 134
Refactoring a reference architecture for your needs 135
Creating your own reference architecture 135
Tactics 136
Externally developed software 136
Buy or build? 137
Advantages/disadvantages of building 137
Advantages/disadvantages of buying 138
Researching external software 138
Should I use open source software (OSS)? 139
Advantages of using open source software 140
Disadvantages of using open source software 140
Documenting the software architecture design 141
Sketching the architecture design 141
Documenting the design rationale 142
Design rationale for design evaluation 143
Design rationale for design verification 143
Design rationale for design knowledge transfer 143
Design rationale for design communication 144
Design rationale for design maintenance 144
Design rationale for design documentation 144
Design rationale for design reuse 144
Using a systematic approach to software architecture design 145
A general model of software architecture design 146
Architectural analysis 146
Architectural synthesis 147
Architectural evaluation 147
Architecture design is an iterative process 147
Selecting an architecture design process 148
Attribute-driven design (ADD) 149
Step 1 – Reviewing inputs 151
Step 2 – Establishing the iteration goal and selecting inputs to be considered in the iteration 151
Step 3 – Choosing one or more elements of the system to refine 152
Step 4 – Choosing one or more design concepts that satisfy the inputs considered in the iteration 152
Step 5 – Instantiating architectural elements, allocating responsibilities, and defining interfaces 152
Step 6 – Sketching views and recording design decisions 153
Step 7 – Performing analysis of current design and reviewing the iteration goal and design objectives 153
Step 8 – Iterating if necessary 153
Microsoft's technique for architecture and design 154
Step 1 – Identifying architecture objectives 154

Step 2 – Identifying key scenarios 155
Step 3 – Creating application overview 155
Determining your application type 156
Identifying your deployment constraints 156
Identifying important architecture design styles 156
Determining relevant technologies 156
Step 4 – Identifying key issues 157
Step 5 – Defining candidate solutions 157
Architecture-centric design method (ACDM) 157
Step 1 – Discovering architectural drivers 159
Step 2 – Establishing project scope 159
Step 3 – Creating notional architecture 159
Step 4 – Architectural review 159
Step 5 – Production go/no-go 160
Step 6 – Experiment planning 160
Step 7 – Experimenting with and refining the architecture 160
Production planning and production 161
Architecture development method (ADM) 161
The Open Group Architecture Framework (TOGAF) 161
TOGAF architecture domains 162
TOGAF documentation 162
Phases of the ADM 163
Phase A – Architecture vision 164
Phase B – Business architecture 164
Phase C – Information systems architectures 165
Phase D – Technology architecture 165
Phase E – Opportunities and solutions 165
Phase F – Migration planning 166
Phase G – Implementation governance 166
Phase H – Architecture change management 166
Tracking the progress of the software architecture's design 166
Using a backlog to track the architecture design progress 167
Prioritizing the backlog 167
DIVE criteria 168
Dependencies 168
Insure against risks 168
Business value 168
Estimated effort 168
Active and dynamic architecture backlogs 168
Summary 169
Chapter 6: Software Development Principles and Practices 171
Designing orthogonal software systems 172
Loose coupling 172
Types of coupling 173
Content coupling 173
Common coupling 173
External coupling 174

Control coupling 174
Stamp coupling (data-structured coupling) 174
Data coupling 174
Message coupling 175
No coupling 175
The Law of Demeter (LoD) / principle of least knowledge 175
Designing for loose coupling 175
High cohesion 176
Types of cohesion 176
Coincidental cohesion 176
Logical cohesion 177
Temporal cohesion 177
Procedural cohesion 177
Communicational cohesion 178
Sequential cohesion 178
Functional cohesion 178
Designing for high cohesion 179
Minimizing complexity 179
KISS principle – "Keep It Simple, Stupid" 180
Origin of KISS 180
Applying KISS to software 181
Don't make it overly simple 181
DRY – "Don't Repeat Yourself" 182
Copy-and-paste programming 182
Magic strings 182
How to avoid duplication 184
Don't make things overly DRY 184
Information hiding 184
Reasons for information hiding 185
What needs to be exposed/hidden? 185
YAGNI – "You Aren't Gonna Need It" 186
Avoid over-engineering a solution 186
Situations where YAGNI doesn't apply 186
Separation of Concerns (SoC) 187
Following SOLID design principles 188
Single Responsibility Principle (SRP) 188
Open/Closed Principle (OCP) 191
Liskov Substitution Principle (LSP) 193
Interface Segregation Principle (ISP) 195
Dependency Inversion Principle (DIP) 198
Inversion of Control (IoC) 199
Dependency Injection (DI) 200
Benefits of DI 200
DI patterns 201
DI containers 204
Helping your team succeed 205
Unit testing 205
What are unit tests? 205
Benefits of unit testing 206
Properties of a good unit test 206

Atomic 206
Deterministic 207
Automated and repeatable 207
Isolated and independent 207
Easy to set up and implement 207
Fast 207
The AAA pattern 208
Arrange 208
Act 208
Assert 208
Naming conventions for unit tests 208
Unit test class names 209
Unit test method names 209
Code coverage for unit tests 209
Keeping unit tests up to date 210
Setting up development environments 210
Providing a README file 211
Pair programming 212
Benefits of pair programming 212
Using pair programming when it is needed 213
Reviewing deliverables 213
Code reviews 213
Formal inspections 214
Roles for a formal inspection 215
Inspection meeting and follow-up 215
Walkthroughs 216
Summary 216
Chapter 7: Software Architecture Patterns 217
Software architecture patterns 218
Using software architecture patterns 218
Overusing architecture patterns 219
Understanding the difference between architecture styles and architecture patterns 219
Layered architecture 220
Open versus closed layers 220
Tiers versus layers 221
Advantages of layered architectures 222
Disadvantages of layered architectures 223
Client-server architecture (two-tier architecture) 224
Using stored procedures for application logic 225
N-tier architecture 226
Presentation tier 227
Business tier 227
Data tier 228
Event-driven architecture 228
Event channels 228
Message queues 229
The point-to-point channel pattern 229
Message topics 229

The publish-subscribe pattern 229
Event-driven architecture topologies 230
The mediator topology 230
Event mediator implementations 231
The broker topology 231
Event processing styles 232
Simple event processing (SEP) 233
Event stream processing (ESP) 233
Complex event processing (CEP) 233
Types of event-driven functionality 234
Event notification 234
Event-carried state transfer 234
Event-sourcing 235
The Model-View-Controller pattern 236
Model 237
View 237
Controller 238
Advantages of the MVC pattern 238
Disadvantages of the MVC pattern 238
The Model-View-Presenter pattern 239
Model 239
View 239
Presenter 240
The Model-View-ViewModel pattern 240
Model 241
View 241
ViewModel 241
The Command Query Responsibility Segregation pattern 242
The query model and the command model 243
Using event-sourcing with CQRS 244
Advantages of CQRS 244
Disadvantages of CQRS 245
Service-oriented architecture 245
What makes SOA different from other distributed solutions? 246
Benefits of using a SOA 246
Increases alignment between business and technology 247
Promotes federation within an organization 247
Allows for vendor diversity 247
Increases intrinsic interoperability 248
Works well with agile development methodologies 248
Cost-benefit analysis of SOA 248
Challenges with SOA 249
Key principles for service orientation 250
Standardized service contract 250
Service loose coupling 250
Service abstraction 251
Service reusability 251

Service autonomy 251
Service statelessness 252
Service discoverability 252
Service composability 252
SOA delivery strategies 252
The top-down strategy 253
The bottom-up strategy 253
The agile strategy 254
Service-oriented analysis 254
Defining business automation requirements 254
Identifying existing automation systems 255
Modeling candidate services 255
Service layers and service models 255
Task service 257
Entity service 257
Utility service 257
Service-oriented design 257
Service interface design 258
Service interface granularity 259
Service registries 259
Service descriptions 260
Structuring namespaces 261
Orchestration and choreography 262
Summary 263
Chapter 8: Architecting Modern Applications 265
Monolithic architecture 265
Benefits of a monolithic architecture 266
Drawbacks of a monolithic architecture 267
Microservice architecture 268
SOA done right 268
Characteristics of microservice architecture 269
Small, focused services 269
Well-defined service interfaces 270
Autonomous and independently deployable services 270
Independent data storage 270
Better fault isolation 271
Communicating with lightweight message protocols 271
Designing polyglot microservices 272
Polyglot programming 273
Polyglot persistence 273
Using too many technologies 274
Considering service granularity 274
Nanoservices 274
Sharing dependencies between microservices 275
Stateless versus stateful microservices 276
Service discovery 276
Using a service registry 276
Self-registration pattern 277

Third-party registration pattern 277
Types of service discovery 278
Client-side discovery pattern 279
Server-side discovery pattern 279
Using microservices is not for everyone 280
Serverless architecture 281
Function as a Service (FaaS) 282
Backend as a Service (BaaS) 283
Advantages of serverless architectures 284
Cost savings 284
Scalable and flexible 284
Focus on building your core products 284
Polyglot development 285
Disadvantages of serverless architectures 285
Difficulties with debugging and monitoring 285
Multitenancy issues 285
Vendor lock-in 286
Complexity of designing many functions 286
Not as many runtime optimizations 286
Still immature 286
Taking a hybrid approach to serverless 287
Function deployment 287
Function invocation 288
Synchronous request 288
Asynchronous request (message queue) 289
Message stream 289
Batch job 290
Cloud-native applications 290
Reasons to move to the cloud 291
Reducing costs 291
Greater flexibility and scalability 291
Automatic updates 291
Disaster recovery 292
What are cloud-native applications? 292
Containerized 292
Dynamically orchestrated 293
Microservices-oriented 294
No downtime 294
Continuous delivery 294
Support for a variety of devices 295
Twelve-factor apps 295
Codebase 296
Dependencies 296
Configuration 297
Backing services 297
Build/Release/Run 297
Processes 298
Port binding 298
Concurrency 299

Disposability 299
Development/production parity 299
Logs 300
Administrative processes 300
Summary 301
Chapter 9: Cross-Cutting Concerns 303
Cross-cutting concerns 304
General guidelines for cross-cutting concerns 305
Identifying cross-cutting concerns 305
Using open-source and third-party solutions 305
Maintaining consistency 305
Avoiding scattered solutions 306
Avoiding tangled solutions 306
Implementing cross-cutting concerns 307
Using dependency injection (DI) 307
Using the decorator pattern 308
Aspect-oriented programming 312
Types of advice 313
Weaving 314
Compile-time weaving 315
Runtime weaving 315
Types of cross-cutting concerns 316
Caching 316
Configuration management 317
Auditing 318
Security 318
Exception management 318
Logging 319
Understanding log levels 319
Routing log entries 320
Using Elastic Stack 321
Elasticsearch 321
Logstash 322
Kibana 322
Beats 323
Cross-cutting concerns for microservices 323
Leveraging a microservice chassis 323
Using the sidecar pattern 324
Summary 326
Chapter 10: Performance Considerations 327
The importance of performance 327
Performance affects user experience 328
Bounce rate 328
Conversion rate 328
Performance is a requirement 329

Page speed affects search rankings 329

Defining performance terminology 329

Latency 330

Throughput 330

Bandwidth 330

Processing time 331

Response time 331

Workload 331

Utilization 331

Taking a systematic approach to performance improvement 332

Profiling an application 332

Instrumentation 333

Statistical profilers 333

Analyzing the results 334

Implementing changes 334

Monitoring results 335

Server-side caching 335

Caching data in distributed applications 336

Using a private caching strategy 336

Using a shared caching strategy 336

Priming the cache 337

Invalidating cached data 337

Expiring data 337

Evicting data 337

Cache usage patterns 338

Cache-aside pattern 338

Read-through pattern 338

Write-through pattern 339

Write-behind pattern 339

Improving web application performance 339

Leveraging HTTP caching 339

Using a validation token 340

Specifying cache-control directives 340

Taking advantage of compression 341

File compression 341

Lossless compression 342

Lossy compression 342

Content-encoding (end-to-end) compression 343

Minifying resources 343

Bundling resources 344

Using HTTP/2 347

Multiplexing 348

Server push 348

Header compression 349

Implementing HTTP/2 349

Using content delivery networks (CDNs) 350

Optimizing web fonts 350

Optimizing the critical rendering path 352
Database performance 353
Designing an efficient database schema 353
Normalizing a database 353
Denormalizing a database 354
Identifying primary and foreign keys 354
Selecting the most appropriate data types 355
Using database indexes 355
Primary/clustered indexes 355
Secondary/non-clustered indexes 356
Having too many indexes 356
Scaling up and out 357
Database concurrency 358
Database transactions 358
Optimistic versus pessimistic concurrency control 358
CAP theorem 359
ACID model 360
Atomicity 360
Consistency 360
Isolation 360
Durability 361
BASE model 361
Basic availability 361
Soft state 361
Eventual consistency 361
Summary 362
Chapter 11: Security Considerations 363
Securing software systems 364
The three states of information 364
The CIA triad 365
Confidentiality 365
Integrity 365
Availability 366
Threat modeling 366
Decomposing an application 367
Identifying and categorizing potential threats 367
STRIDE threat model 368
Spoofing identity 368
Tampering with data 368
Repudiation 368
Information disclosure 369
Denial-of-service 369
Elevation of Privilege 369
Prioritizing potential threats 369
DREAD risk assessment model 370
Damage potential 370
Reproducibility 370
Exploitability 371
Affected users 371

Discoverability 371
Responses to threats 371
Avoiding the risk 372
Transferring the risk 372
Accepting the risk 372
Mitigating the risk 373
Types of security control 373
Physical security controls 373
Administrative controls 374
Technical security controls 375
Prevention 375
Detection 376
Response 376
Secure by design 376
Minimizing the attack surface 377
Defense in depth 377
Principle of least privilege (PoLP) 377
Avoiding security by obscurity 378
Keep software designs simple 378
Secure by default 378
Default deny 379
Validating input 379
Secure the weakest link 379
Security must be usable 379
Fail securely 380
Cryptography 380
Encryption 380
Symmetric (secret key) encryption 381
Asymmetric (public key) encryption 381
Cryptographic hash functions 381
Identity and access management (IAM) 383
Authentication 383
What is multi-factor authentication (MFA)? 384
Authorization 384
Storing plaintext passwords 385
Storing encrypted passwords 385
Storing hashed passwords 385
Using domain authentication 386
Implementing a centralized identity provider (IdP) 386
OAuth 2/OpenID Connect (OIDC) 387
OAuth 2 roles 387
JSON web token (JWT) 388
Header 389
Payload 389
Signature 390
Authorizing with the authorization server 390
Most common web application security risks 391
Injection 391

Broken authentication 391
Sensitive data exposure 392
XML external entity (XXE) attack 392
Broken access control 394
Security misconfiguration 394
Cross-site scripting (XSS) 395
Insecure deserialization 395
Using components with known vulnerable components 396
Insufficient logging and monitoring 396
Unvalidated redirects and forwards 397
Summary 397
Chapter 12: Documenting and Reviewing Software Architectures 399
Uses of software architecture documentation 400
Communicating your architecture to others 400
Assisting the development team 401
Educates team members 401
Providing input for software architecture reviews 401
Allowing for the reuse of architectural knowledge 402
Help the software architect 402
Creating architecture descriptions (ADs) 403
Software architecture views 403
Software architecture notations 404
Informal software architecture notations 404
Semiformal software architecture notations 405
Formal software architecture notations 405
Including design rationales 406
Overview of the Unified Modeling Language (UML) 407
Types of modeling 407
Class diagrams 407
Visibility 408
Association 408
Aggregation 409
Composition 410
Multiplicity 410
Dependency 411
Generalization/specialization 411
Realization 412
Component diagrams 413
Package diagrams 415
Deployment diagrams 416
Use case diagrams 417
Sequence diagrams 419
Lifeline 420
Activation boxes 420
Messages 420
Loops 421
Optional flows 422
Alternative flows 422
Activity diagrams 423

Start/end nodes 424
Actions/Control flow 425
Decision/merge nodes 425
Fork/join nodes 426
Reviewing software architectures 426
Software architecture analysis method (SAAM) 427
Scenario-based analysis of software architecture 427
SAAM steps 428
Step 1 – Develop scenarios 428
Step 2 – Describe the architecture 428
Step 3 – Classify and prioritize scenarios 428
Step 4 – Evaluate scenarios 429
Step 5 – Assess scenario interaction 429
Step 6 – Create an overall evaluation 429
Architecture tradeoff analysis method (ATAM) 430
ATAM participant roles 430
ATAM phases 430
Phase 0 – Partnership and preparation 431
Phase 1 – Evaluation 431
Phase 2 – Evaluation (continued) 434
Phase 3 – Follow-up 435
Active design review (ADR) 435
ADR steps 436
Step 1 – Prepare the documentation for review 436
Step 2 – Identify the specialized reviews 437
Step 3 – Identify the reviewers needed 437
Step 4 – Design the questionnaires 437
Step 5 – Conduct the review 437
Active reviews of intermediate designs (ARID) 438
ARID participant roles 438
ARID phases 438
Phase 1 – Pre-meeting 439
Phase 2 – Review meeting 440
Summary 441
Chapter 13: DevOps and Software Architecture 443
DevOps 444
CALMS 444
Culture 445
Automation 446
Lean 447
Measurement 448
Sharing 448
Why DevOps? 448
DevOps toolchain 449
DevOps practices 451
Continuous integration (CI) 452
Automated builds 452
Software versioning 453
Automated testing 454
Continuous delivery (CD) 455

Continuous deployment 455

Architecting for DevOps 456

Important quality attributes for DevOps 456

Some architecture patterns complement DevOps 458

Deploying to the cloud 458

Cloud types 459

Public cloud 459

Private cloud 459

Hybrid cloud 460

Cloud models 461

Infrastructure as a service (IaaS) 462

Containers as a Service (CaaS) 463

Platform as a Service (PaaS) 465

Serverless/Function as a Service (FaaS) 466

Software as a Service (SaaS) 466

Summary 467

Chapter 14: Architecting Legacy Applications 469

Legacy applications 469

Issues with legacy applications 470

Why are legacy applications used? 471

More than just code 472

Refactoring legacy applications 472

Making legacy code testable 473

Benefits of unit testing 473

Refactoring for unit tests 474

Where to start writing tests? 474

Removing redundant code 475

Unreachable code 475

Dead code 476

Commented-out code 476

Duplicate code 476

Using tools to refactor 477

Making small, incremental changes 477

Transforming monoliths to microservices 477

Migrating to the cloud 478

The 6 R's 479

Remove (or retire) 479

Retain 479

Replatform 480

Rehost 480

Repurchase 480

Refactor (or re-architect) 480

Moving to an agile approach 481

Modernizing build and deployment processes 481

Automating the build and deployment processes 482

Practicing continuous integration (CI) 482

Practicing continuous delivery (CD) 483
Updating the build tools 483
Integrating with legacy applications 483
Confirming the need for integration 484
Determining the type of integration 484
Sharing functionality between systems 485
Performing data integration 485
Summary 485
Chapter 15: The Soft Skills of Software Architects 487
Soft skills 487
Communication 488
Communicating the architecture 488
Communicating about quality attributes 489
Communicating expectations 489
The 7 Cs of communication 489
Clarity 490
Conciseness 490
Concreteness 491
Courteousness 491
Consideration 491
Correctness 492
Completeness 492
Listening skills 493
Hearing is not listening 493
Showing empathy 493
Tips for effective listening 493
Giving presentations 494
The 4 Ps of presentations 495
Plan 495
Prepare 495
Practice 496
Present 497
Leadership 497
Getting others to follow you 497
Dealing with challenges 498
Being a technical leader 498
Taking responsibility 498
Focusing on others 499
Delegating tasks 499
Driving change 500
Communication and leadership 500
Mentoring others 501
Leading by example 501
Depending on others 502
Negotiation 502
How negotiation skills may be used 503

Informal/formal negotiations 503

Working with remote resources 505

Benefits of using remote resources 505

Challenges when using remote resources 505

Communication 506

Cultural differences 506

Impromptu meetings 506

New employee onboarding 506

Work quality 507

Confidential company data 507

Summary 508

Chapter 16: Evolutionary Architecture 509

Change is inevitable 509

Reasons for change 510

Expecting change 510

Lehman's laws of software evolution 511

Lehman's software categories 511

S-type systems 511

P-type systems 512

E-type systems 512

The laws 512

Law I – Continuing change 513

Law II – Increasing complexity 513

Law III – Self-regulation 513

Law IV – Conservation of organizational stability 514

Law V – Conservation of familiarity 514

Law VI – Continuing growth 514

Law VII – Declining quality 514

Law VIII – Feedback system 515

Designing evolutionary architectures 515

Making guided architectural changes 516

Fitness functions 516

Categories of fitness functions 517

Examples of fitness functions 518

Making incremental changes 519

Architectural changes across multiple dimensions 520

Loosely coupled architectures 520

Designing evolvable APIs 521

Applying Postel's Law to APIs 522

Using standards in your software system 522

Last responsible moment (LRM) 523

Summary 523

Chapter 17: Becoming a Better Software Architect 525

Practicing continuous learning 526

Improving the breadth and depth of your knowledge 526

Avoiding the law of the instrument 527

Finding the time for learning 527
Ways to keep your skills sharp 528
Participating in open source projects 529
Creating your own open source project 529
Writing your own blog 530
Increasing your visibility 531
Starting your own blog 531
Things to avoid 532
Spending time teaching others 532
Finding opportunities to teach 533
Being a mentor 533
Trying new technologies 534
Continuing to write code 535
Assigning yourself coding tasks 535
Working on your own project 535
Reading code 536
Attending user groups and conferences 536
Presenting at a user group or conference 537
Meeting new people 538
Taking responsibility for your work 538
Attending to your well-being 539
Being proud of your work 539
Summary 539
Other Books You May Enjoy 541
Index 545