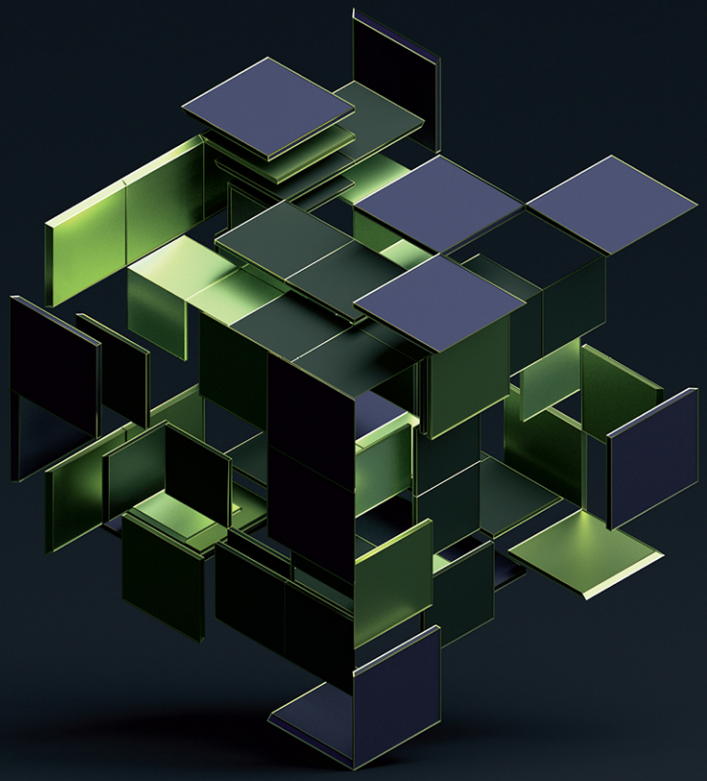
ESTABLISHING SRE FOUNDATIONS

A Step-by-Step Guide to Introducing Site Reliability Engineering in Software Delivery Organizations



VLADYSLAV UKIS

Foreword by DAVID FARLEY

Praise for *Establishing SRE Foundations*

"Many enterprises today face the challenge of establishing modern operations for their SaaS offerings. This book provides a proven step-by-step guide for how this can be done from scratch using Google's SRE methodology. From achieving organizational buy-in to laying down the basic SRE foundations, establishing incident response and implementing a suitable organizational structure—the book contains a wealth of advice for development, operations, and leadership teams!"

—Dr. Peter Schardt, Chief Technology Officer at Siemens Healthcare GmbH

"*Establishing SRE Foundations* is a great introductory guide for anyone new to understanding and implementing Site Reliability Engineering (SRE) in their organization. Vlad creates a solid platform for anyone wishing to understand the SRE approach to building reliability into software services. As well as practical advice on implementing techniques such as SLIs and SLOs, Vlad goes into detail on how to achieve buy-in for SRE adoption and how to modify your organizational setup, rooted in his own experiences of working in a large organization. Those experiences are sorely lacking elsewhere in SRE literature, and when I'm asked in the future about SRE, I'll be referring people to this excellent book."

—Steve Smith, author of *Measuring Continuous Delivery* (2020)

"I very much enjoyed reading this book, even in its early forms. Vlad treats the topic of SRE methodically and in great detail; if you have ever been wondering whether or not someone else has come across your particular issue in an SRE implementation, this book can answer that question and probably has an actionable solution as well. Destined to become a constantly referenced handbook by all those involved in SRE change projects."

—Niall Murphy, co-author of *Site Reliability Engineering* (2016) and *The Site Reliability Handbook* (2018)

"There are an overwhelming number of blogs, books, podcasts, and ad hoc opinions covering the nitty-gritty of SRE toolchains and technology choices. That being said, SRE initiatives rarely fail for technological reasons—they fail for structural or organizational reasons. In *Establishing SRE Foundations*, Dr. Ukis has given us all a detailed, accessible, and actionable blueprint for the structures and practices of a successful SRE organization. It is an excellent book and one I would recommend to anyone looking to establish a scaled-out SRE practice in a complex environment."

—Ben Sigelman, co-founder of Lightstep

"*Establishing SRE Foundations* provides far and away the clearest, most comprehensive, and most actionable roadmap I have seen for driving, scaling, and sustaining SRE in an engineering organization. I cannot recommend it highly enough!"

—Randy Shoup, eBay Chief Architect and former Google Engineering leader

"*Establishing SRE Foundations* is a comprehensive guide for anyone looking to take their software operations to the next level. If you are a beginner, you will learn why SRE is a great methodology for improving operations, what the challenges of introducing SRE are, how to achieve organizational buy-in for SRE, how to lay the foundation for SRE in your teams, and how to drive continuous improvement. If you are an experienced practitioner, you will learn how to set up an error budget policy, enable error budget–based decision-making, and implement a suitable organizational structure. I think the content of the book is spot on and highly recommend it!"

—Vitor dos Reis, Director of Software Engineering at Delivery Hero

"Vlad offers a detailed and comprehensive overview of the transformation to SRE. He covers assessment, organizational structures, technical implementation, communication, and continuation. This book is a clear roadmap for any organization starting or progressing their SRE journey, replete with what to consider, options available, and real-world examples. If you are thinking about starting the SRE Journey, have found yourself stalled along the way, or are looking for more ideas to help you continue the journey successfully, then buy this book."

—Doc Norton, Change Catalyst, OnBelay Consulting

Establishing SRE Foundations

This page intentionally left blank

Establishing SRE Foundations

A Step-by-Step Guide to Introducing Site Reliability Engineering in Software Delivery Organizations

Vladyslav Ukis

Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2022937194

Cover image: VAlex/Shutterstock

ISBN-13: 978-0-13-742460-3
ISBN-10: 0-13-742460-4

ScoutAutomatedPrintCode

Pearson's Commitment to Diversity, Equity, and Inclusion

Pearson is dedicated to creating bias-free content that reflects the diversity of all learners. We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, socioeconomic status, ability, age, sexual orientation, and religious or political beliefs.

Education is a powerful force for equity and change in our world. It has the potential to deliver opportunities that improve lives and enable economic mobility. As we work with authors to create content for every product and service, we acknowledge our responsibility to demonstrate inclusivity and incorporate diverse scholarship so that everyone can achieve their potential through learning. As the world's leading learning company, we have a duty to help drive change and live up to our purpose to help more people create a better life for themselves and to create a better world.

Our ambition is to purposefully contribute to a world where:

- Everyone has an equitable and lifelong opportunity to succeed through learning.
- Our educational products and services are inclusive and represent the rich diversity of learners.
- Our educational content accurately reflects the histories and experiences of the learners we serve.
- Our educational content prompts deeper discussions with learners and motivates them to expand their own learning (and worldview).

While we work hard to present unbiased content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.

- Please contact us with concerns about any potential bias at https://www.pearson.com/report-bias.html.

This page intentionally left blank

To my wonderful wife, Lina, daughter, Annika,
and son, Jonas

This page intentionally left blank

Contents

Foreword xxi
Preface xxv
Acknowledgments xxix
About the Author xxxiii

Part I Foundations 1

Chapter 1 Introduction to SRE 3
1.1 Why SRE? 3
1.1.1 ITIL 3
1.1.2 COBIT 4
1.1.3 Modeling 5
1.1.4 DevOps 6
1.1.5 SRE 7
1.1.6 Comparison 8
1.2 Alignment Using SRE 13
1.3 Why Does SRE Work? 17
1.4 Summary 19

Chapter 2 The Challenge 21
2.1 Misalignment 22
2.2 Collective Ownership 23
2.3 Ownership Using SRE 25
2.3.1 Product Development 25
2.3.2 Product Operations 28
2.3.3 Product Management 32
2.3.4 Benefits and Costs 36
2.4 The Challenge Statement 38
2.5 Coaching 39
2.6 Summary 41

Chapter 3 SRE Basic Concepts 43
3.1 Service Level Indicators 43
3.2 Service Level Objectives 45
3.3 Error Budgets 47
3.3.1 Availability Error Budget Example 49
3.3.2 Error Budget of Zero 50
3.3.3 Latency Error Budget Example 52
3.4 Error Budget Policies 53
3.5 SRE Concept Pyramid 55
3.6 Alignment Using the SRE Concept Pyramid 59
3.7 Summary 63

Chapter 4 Assessing the Status Quo 65
4.1 Where Is the Organization? 65
4.1.1 Organizational Structure 65
4.1.2 Organizational Alignment 67
4.1.3 Formal and Informal Leadership 68
4.2 Where Are the People? 69
4.3 Where Is the Tech? 71
4.4 Where Is the Culture? 74
4.4.1 Is There High Cooperation? 75
4.4.2 Are Messengers Trained? 77
4.4.3 Are Risks Shared? 77
4.4.4 Is Bridging Encouraged? 78
4.4.5 Does Failure Lead to Inquiry? 78
4.4.6 Is Novelty Implemented? 78
4.5 Where Is the Process? 79
4.6 SRE Maturity Model 81
4.7 Posing Hypotheses 81
4.8 Summary 86

Part II Running the Transformation 87

Chapter 5 Achieving Organizational Buy-In 89
5.1 Getting People Behind SRE 89
5.2 SRE Marketing Funnel 92
5.2.1 Awareness 93
5.2.2 Interest 93

5.2.3 Understanding 95
5.2.4 Agreement 95
5.2.5 Engagement 96
5.3 SRE Coaches 96
5.3.1 Qualities 97
5.3.2 Responsibilities 98
5.4 Top-Down Buy-In 99
5.4.1 Stakeholder Chart 100
5.4.2 Engaging the Head of Development 103
5.4.3 Engaging the Head of Operations 107
5.4.4 Engaging the Head of Product Management 110
5.4.5 Achieving Joint Buy-In 112
5.4.6 Getting SRE into the Portfolio 114
5.5 Bottom-Up Buy-In 117
5.5.1 Engaging the Operations Teams 117
5.5.2 Engaging the Development Teams 119
5.6 Lateral Buy-In 122
5.7 Buy-In Staggering 123
5.8 Team Coaching 124
5.9 Traversing the Organization 126
5.9.1 Grouping the Organization 126
5.9.2 Traversing the Organization Versus SRE Infrastructure Demand 127
5.9.3 Team Engagements Over Time 129
5.10 Organizational Coaching 131
5.11 Summary 133

Chapter 6 Laying Down the Foundations 135

6.1 Introductory Talks by Team 135
6.2 Conveying the Basics 136
6.2.1 SLO as a Contract 137
6.2.2 SLO as a Proxy Measure of Customer Happiness 138
6.2.3 User Personas 138
6.2.4 User Story Mapping 140
6.2.5 Motivation to Fix SLO Breaches 142
6.2.6 SLOs Are Not About Technicalities 144
6.2.7 Causes of SLO Breaches 145
6.2.8 On Call for SLO Breaches 146

6.3 SLI Standardization 147
6.3.1 Application Performance Management Facility 149
6.3.2 Availability 150
6.3.3 Latency 151
6.3.4 Prioritization 152
6.4 Enabling Logging 154
6.5 Teaching the Log Query Language 156
6.6 Defining Initial SLOs 157
6.6.1 What Makes a Good SLO? 157
6.6.2 Iterating on an SLO 159
6.6.3 Revising SLOs 162
6.7 Default SLOs 163
6.8 Providing Basic Infrastructure 164
6.8.1 Dashboards 165
6.8.2 Alert Content 166
6.9 Engaging Champions 167
6.10 Dealing with Detractors 168
6.10.1 Issues with the Cause 168
6.10.2 Issues with Alerting 168
6.10.3 Issues with Tooling 169
6.10.4 Issues with Product Owner Involvement 170
6.10.5 Issues with Team Motivation 170
6.11 Creating Documentation 171
6.12 Broadcast Success 172
6.13 Summary 174

Chapter 7 Reacting to Alerts on SLO Breaches 175

7.1 Environment Selection 175
7.2 Responsibilities 177
7.2.1 Dev Versus Ops Responsibilities 177
7.2.2 Operational Responsibilities 178
7.2.3 Splitting Operational Responsibilities 179
7.3 Ways of Working 180
7.3.1 Interruption-Based Working Mode 181
7.3.2 Focus-Based Working Mode 185

7.4 Setting Up On-Call Rotations 185
7.4.1 Initial Rotation Period 186
7.4.2 One Person On Call 186
7.4.3 Two People On Call 187
7.4.4 Three People On Call 187
7.5 On-Call Management Tools 188
7.5.1 Posting SLO Breaches 188
7.5.2 Scheduling 190
7.5.3 Professional On-Call Management Tools 191
7.6 Out-of-Hours On-Call 193
7.6.1 Using Availability Targets and Product Demand 194
7.6.2 Trade-offs 194
7.7 Systematic Knowledge Sharing 196
7.7.1 Knowledge-Sharing Needs 198
7.7.2 Knowledge-Sharing Pyramid 199
7.7.3 On-Call Training 201
7.7.4 Runbooks 203
7.7.5 Internal Stack Overflow 205
7.7.6 SRE Community of Practice 206
7.8 Broadcast Success 208
7.9 Summary 209

Chapter 8 Implementing Alert Dispatching 211

8.1 Alert Escalation 212
8.2 Defining an Alert Escalation Policy 214
8.3 Defining Stakeholder Groups 216
8.4 Triggering Stakeholder Notifications 218
8.5 Defining Stakeholder Rings 219
8.6 Defining Effective Stakeholder Notifications 222
8.7 Getting the Stakeholders Subscribed 225
8.7.1 Subscribing Using the On-Call Management Tool 225
8.7.2 Subscribing Using Other Means 226
8.8 Broadcast Success 226
8.9 Summary 227

Chapter 9 Implementing Incident Response 229
9.1 Incident Response Foundations 229
9.2 Incident Priorities 230
9.2.1 SLO Breaches Versus Incidents 232
9.2.2 Changing Incident Priority During an Incident 233
9.2.3 Defining Generic Incident Priorities 234
9.2.4 Mapping SLOs to Incident Priorities 237
9.2.5 Mapping Error Budgets to Incident Priorities 239
9.2.6 Mapping Resource-Based Alerts to Incident Priorities 240
9.2.7 Uncovering New Use Cases for Incident Priorities 242
9.2.8 Adjusting Incident Priorities Based on Stakeholder Feedback 242
9.2.9 Extending the SLO Definition Process 244
9.2.10 Infrastructure 245
9.2.11 Deduplication 246
9.3 Complex Incident Coordination 248
9.3.1 What Is a Complex Incident? 248
9.3.2 Existing Incident Coordination Systems 249
9.3.3 Incident Classification 250
9.3.4 Defining Generic Incident Severities 251
9.3.5 Social Dimension of Incident Classification 252
9.3.6 Incident Priority Versus Incident Severity 253
9.3.7 Defining Roles 254
9.3.8 Roles Required by Incident Severity 257
9.3.9 Roles On Call 257
9.3.10 Incident Response Process Evaluation 258
9.3.11 Incident Response Process Dynamics 260
9.3.12 Incident Response Team Well-Being 262
9.4 Incident Postmortems 268
9.5 Effective Postmortem Criteria 269
9.5.1 Initiating a Postmortem 271
9.5.2 Postmortem Lifecycle 272
9.5.3 Before the Postmortem 273
9.5.4 During the Postmortem 276
9.5.5 After the Postmortem 283
9.5.6 Analyzing the Postmortem Process 283

9.5.7 Postmortem Template 289
9.5.8 Facilitating Learning from Postmortems 291
9.5.9 Successful Postmortem Practice 291
9.5.10 Example Postmortems 292
9.6 Mashing Up the Tools 294
9.6.1 Connecting to the On-Call Management Tool 294
9.6.2 Connections Among Other Tools 296
9.6.3 Mobile Integrations 297
9.6.4 Example Tool Landscapes 298
9.7 Service Status Broadcast 298
9.8 Documenting the Incident Response Process 301
9.9 Broadcast Success 302
9.10 Summary 303

Chapter 10 Setting Up an Error Budget Policy 305

10.1 Motivation 305
10.2 Terminology 307
10.3 Error Budget Policy Structure 308
10.4 Error Budget Policy Conditions 309
10.5 Error Budget Policy Consequences 311
10.6 Error Budget Policy Governance 312
10.7 Extending the Error Budget Policy 314
10.8 Agreeing to the Error Budget Policy 318
10.9 Storing the Error Budget Policy 319
10.10 Enacting the Error Budget Policy 320
10.11 Reviewing the Error Budget Policy 321
10.12 Related Concepts 322
10.13 Summary 324

Chapter 11 Enabling Error Budget–Based Decision–Making 325

11.1 Reliability Decision-Making Taxonomy 325
11.2 Implementing SRE Indicators 330
11.2.1 Dimensions of SRE Indicators 330
11.2.2 “SLOs by Service” Indicator 330
11.2.3 SLO Adherence Indicator 332
11.2.4 SLO Error Budget Depletion Indicator 333

11.2.5 Premature SLO Error Budget Exhaustion Indicator 339
11.2.6 "SLAs by Service" Indicator 343
11.2.7 SLA Error Budget Depletion Indicator 345
11.2.8 SLA Adherence Indicator 348
11.2.9 Customer Support Ticket Trend Indicator 349
11.2.10 "On-Call Rotations by Team" Indicator 353
11.2.11 Incident Time to Recovery Trend Indicator 355
11.2.12 Least Available Service Endpoints Indicator 356
11.2.13 Slowest Service Endpoints Indicator 358
11.3 Process Indicators, Not People KPIs 359
11.4 Decisions Versus Indicators 359
11.5 Decision-Making Workflows 362
11.5.1 API Consumption Decision Workflow 363
11.5.2 Tightening a Dependency's SLO Decision Workflow 366
11.5.3 Features Versus Reliability Prioritization Workflow 368
11.5.4 Setting an SLO Decision Workflow 372
11.5.5 Setting an SLA Decision Workflow 377
11.5.6 Allocating SRE Capacity to a Team Decision Workflow 380
11.5.7 Chaos Engineering Hypotheses Selection Workflow 383
11.6 Summary 388

Chapter 12 Implementing Organizational Structure 391

12.1 SRE Principles Versus Organizational Structure 393
12.2 Who Builds It, Who Runs It? 394
12.2.1 "Who Builds It, Who Runs It?" Spectrum 395
12.2.2 Hybrid Models 396
12.2.3 Reliability Incentives 397
12.2.4 Model Comparison Criteria 400
12.2.5 Model Comparison 402
12.3 You Build It, You Run It 403
12.4 You Build It, You and SRE Run It 406
12.4.1 SRE Team Within the Development Organization 406
12.4.2 SRE Team Within the Operations Organization 409
12.4.3 SRE Team in a Dedicated SRE Organization 410
12.4.4 Comparison 411
12.4.5 SRE Team Incentives, Identity, and Pride 412
12.4.6 SRE Team Head Count and Budget 414
12.4.7 SRE Team Cost Accounting 417
12.4.8 SRE Team KPIs 419

12.5 You Build It, SRE Run It 421
12.5.1 SRE Team Within a Development Organization 421
12.5.2 SRE Team Within an Operations Organization 423
12.5.3 SRE Team in a Dedicated SRE Organization 423
12.6 Cost Optimization 424
12.7 Team Topologies 426
12.7.1 Reporting Lines 427
12.7.2 SRE Identity Triangle 428
12.7.3 Holacracy: No Reporting Lines 431
12.8 Choosing a Model 432
12.8.1 Model Transformation Options 432
12.8.2 Decision Dimensions 434
12.8.3 Reporting Options 435
12.8.4 Positioning the SRE Organization 437
12.8.5 Conveying the Value to Executives 439
12.9 A New Role: SRE 440
12.9.1 Why Is a New Role Needed? 440
12.9.2 Role Definition 443
12.9.3 Role Naming 445
12.9.4 Role Assignment 447
12.9.5 Role Fulfillment 448
12.10 SRE Career Path 450
12.10.1 SRE Role Progressions 451
12.10.2 SRE Role Transitions 453
12.10.3 Cultural Importance 455
12.11 Communicating the Chosen Model 456
12.12 Introducing the Chosen Model 457
12.12.1 Organization Changes 458
12.12.2 Reporting Structure Changes 460
12.12.3 Role Changes 461
12.13 Summary 462

Part III Measuring and Sustaining the Transformation 465

Chapter 13 Measuring the SRE Transformation 467

13.1 Testing Transformation Hypotheses 467
13.2 Outages Not Detected Internally 469
13.3 Services Exhausting Error Budgets Prematurely 470

13.4 Executives' Perceptions 471
13.5 Reliability Perception by Users and Partners 472
13.6 Summary 473

Chapter 14 Sustaining the SRE Movement 475

14.1 Maturing the SRE CoP 475
14.2 SRE Minutes 475
14.3 Availability Newsletter 476
14.4 SRE Column in the Engineering Blog 477
14.5 Promote Long-Form SRE Wiki Articles 477
14.6 SRE Broadcasting 478
14.7 Combining SRE and CD Indicators 479
14.7.1 CD Versus SRE Indicators 481
14.7.2 Bottleneck Analysis 482
14.8 SRE Feedback Loops 483
14.9 New Hypotheses 484
14.10 Providing Learning Opportunities 486
14.11 Supporting SRE Coaches 487
14.12 Summary 489

Chapter 15 The Road Ahead 491

15.1 Service Catalog 492
15.2 SLAs 494
15.3 Regulatory Compliance 494
15.4 SRE Infrastructure 495
15.5 Game Days 496

Appendix Topics for Quick Reference 499

Index 507

Foreword

I first met Vlad Ukis at a QCon conference in London a few years ago. He wanted to recruit me as a consultant to help advise his team at Siemens Healthcare. I worked with the teamplay digital health platform team that Vlad led in Siemens Healthcare over the course of the next year or so, and over that period Vlad and I became friends.

Vlad has done an outstanding job helping the teamplay team, and more broadly Siemens Healthcare to make fantastic progress. The hard-won lessons that he and his team worked through are writ large in the pages of this book.

The teamplay team are applying the advanced, engineering-led, modern version of agile development exemplified by Continuous Delivery, DevOps and SRE, to significant advantage. They demonstrate the applicability of these ideas beyond the bounds of the big web companies that most people tend to think of when we discuss these ideas.

I often see and hear organizations dismiss sometimes important ideas that were popularized by the big web companies with comments like, "Yes, but we aren't <Google, Amazon, Netflix, insert your favorite here>." This is a misreading of why these ideas work in those organizations.

It is not always that the problems in the big web companies are unique. Rather, it is that their scale means that common problems often become limiting more quickly. This means that it becomes essential for them to solve these common problems. These big organizations don't practice Continuous Delivery (CD) and SRE because they are fads. They practice them because they work better than any alternatives that we know and address problems at the heart of all software development.

As an early adopter and promoter of some of these ideas, I think that we have entered a new phase in the evolution of some of these ideas. We are now seeing them being adopted more widely, and to very significant advantage and effect, in all kinds of software development organizations. Automotive, aerospace, telecoms, and medical sectors all have examples of their use. This book makes that clear with an example from a real-world complex software development. It stops people being able to say "SRE is all very well, but we are not Google." It is also a lot more than only that, though.

I think there are very good reasons for the growth of ideas like CD and SRE. Both are true engineering approaches to solving problems. They both try to use measurement and apply scientific style reasoning to solving real-world, practical problems that we all face, whatever the scale of our software development or the nature of the problem. I describe CD as being driven by enabling an experimental approach to software development. SRE is profoundly that too.

I have written about my views on applying engineering thinking to the development of software. I think learning, and evolving, our discipline in this direction is essential to doing a good job. Why does this matter in the context of this book? I think it is important to remember that the "E" in "SRE"

means "Engineering"; it is not just the word "SRE". My preferred definition for software engineering is this:

> Software engineering is the application of an empirical, scientific approach to finding efficient, economic solutions to practical problems in software.

SRE thinking is profoundly grounded in the principles at the heart of this definition. It also adopts that other essential aspect of true engineering: We start off assuming that we will make mistakes.

The world isn't perfect. Our software won't always perform as we hope. Every system fails sometimes. SRE puts this kind of thinking front and center, and forces us, as teams and organizations, to think about how we would like our systems to cope—*How much down time is too much?* and *What shall we do when approaching those limits?*

This book does two things and does both extremely well.

At its heart, this book describes how the engineering approach that underpins SRE provides greater clarity and more effective collaboration between the three main strands of development: People focused primarily on the product, its development, and its operation.

SRE provides the glue between these groups, focusing them on what really matters in a way that is collaborative but also leaves each group with enough clarity to inform independent decision-making in their own sphere.

The techniques and principles of SRE are not only clearly defined here, but also the rationale behind them is explained in a way that will stick. This is not some dry definition, this is practical, usable, understanding.

The second thing that this book does is to describe how to start making changes to apply this kind of thinking, and the techniques of SRE, in a preexisting real-world, complex development organization. This is clearly based on much more than a theoretical understanding or interpretation—these are words from a practitioner.

The teamplay team are not dealing with simple software. Their work cannot be easily dismissed, inaccurate as those dismissals usually are, as just being another simple website or online shop. The teamplay team build real software that matters. Their software helps to save lives of patients in hospitals. It integrates world-class medical devices in hospitals with information systems in the cloud that enable new insights and new ways to help people. They adopt these leading-edge techniques, not because they are fashionable but because they work better than anything else that we know how to do so far.

This book will certainly help you understand what ideas like Service Level Indicators (SLIs), Service Level Objectives (SLOs), and Error Budgets really mean; their relationship to one another; and how to apply them. It explores, in some detail, how to organize effective responses to incidents and how to perform good post-mortems after incidents to reinforce learning. It describes effective organizational structures. This is a wide-ranging book for a wide-ranging topic. For me, though, it goes even beyond that.

I am a long-time practitioner of ideas that are incredibly well aligned and close to these ideas. I have read around this subject for several years and thought that I really understood what it was about. But my understanding is deeper now. I really get it and plan to add more use of SRE ideas to the way that I communicate and explain things in my own work. I thank Vlad for that.

There are nuggets in here that inform my own thinking and help me to develop my understanding of what it takes to apply real engineering thinking to software development.

I already knew that in engineering it is all about trade-offs, but Vlad explains this very clearly with examples in an SRE context and describes what some of the common trade-offs are and how to think about them.

I laughed out loud when I read the now blindingly obvious statement that "If you set your SLO to 100%, that means that features are always second priority." Of course, that is true. I knew this, but now I have better words and better models to express it with.

I was delighted and honored to be asked to write a foreword for this book. I confess that I may have written a foreword for it anyway because Vlad is intelligent, thoughtful, and does really good work, but also because he is my friend. I am doubly delighted that I don't need to do this as a favor to a friend, though. I can whole-heartedly recommend this book without any reservation. This is a very good book on an important topic that helps to move the game forward for our discipline! I hope that you enjoy it as much as I have.

—Dave Farley
Independent Software Development Consultant
Founder and CEO of Continuous Delivery Ltd.

This page intentionally left blank

Preface

This book is based on a site reliability engineering (SRE) transformation journey from a real software delivery organization in the healthcare industry. The organization runs a cloud-based platform for medical applications and services. The platform is deployed in many data centers and the applications on the platform are used in hospitals around the world. Some of the applications are used when patients are in critical condition. It follows, then, that the platform's reliability is of paramount importance.

But what is reliability? How do you measure it? How do you create an environment where development teams are motivated to invest in reliability? These were the questions I grappled with several years ago when the organization struggled to provide a reliable platform for applications and users alike. High-profile customer escalations were common. People were unaligned regarding backlog prioritization of new features versus reliability work. The operations teams were struggling to operate the product. The development teams happily implemented new features but paid very little attention to how the existing features were running in production. Project management plans were impacted greatly by deployment of large numbers of unexpected hotfixes. High-profile customers called the leadership team demanding that the service be restored or that missing features be delivered. Everyone had an opinion on what needed to be done to improve the situation, until the next outage took place, causing new opinions to emerge.

I had attended the QCon London conference for several years. The conference helped me stay abreast of new trends in software development and operations. SRE was one of the topics at the conference. I was aware of its existence but had not started learning about it. At one of the QCons, an entire track was devoted to SRE. I spent a significant amount of time attending sessions in that track. At the end of the conference, it was clear to me that SRE was gaining momentum in the industry.

While traveling back to work from the conference and looking over my notes, I decided that it was the right time for the organization to try SRE in an attempt to improve operations. There was no other structured approach to doing operations that I had come across. What we tried ourselves without SRE did not yield visible improvements. Many companies at the conference reported being successful, whatever that meant, in doing operations using SRE. Getting started seemed to be easy. It would only take a couple of basic indicators, like availability and latency, the definition of acceptable targets for each service, and alerts on when the targets would be broken.

Once I was back at work, I started thinking about how to drive SRE from within my organizational unit. Thinking deeper, I realized I would need engagement from the entire organization. The questions I had in mind were as follows:

- How would I drum up support for SRE in the organization?
- How would I engage the leadership team?

- How would I engage the operations teams?
- How would I engage the development teams? There was a growing number of them, soon to be 20 or more. So, how would I drive SRE in a growing organization, in a way that would scale with the growing number of teams expected to emerge in the future?
- What is SRE at a deeper level?
- Why does it work?
- How could I learn more about SRE?
- How could I learn enough about SRE to explain it to others quickly and easily?
- Is there an alternative to SRE?
- How could I engage with people who had already introduced SRE in their organizations?
- What are the common pitfalls of introducing SRE in an organization and how would I avoid them?

With these questions in mind, a period of soul searching followed. To cut a long story short, we managed to establish SRE as the central discipline in the organization's development and operations departments. Doing so significantly and measurably improved our ability to operate the global platform.

Moreover, the organization is in touch with many teams that build applications on top of the platform. How to operate those applications effectively is a common question from the teams. We now routinely teach SRE as a preferred method of doing operations. The teams introduce it and use the SRE infrastructure we provide.

During the SRE transformation, we got a chance to visit Delivery Hero in Berlin. They were running operations at a world-class level. It was inspiring to learn from them back then. It was even more inspiring to later see our own teams getting close to being world class.

Along the way, many lessons were learned. Introducing SRE at scale to a development organization that had never done operations and to an operations organization that had never enabled others to do operations is a very significant undertaking. It requires deep, long-term engagement with the development teams providing coaching on their individual journeys toward growing maturity in operational capabilities. At the same time, it requires long-term engagement with the operations teams providing coaching on their journey toward becoming an SRE infrastructure framework provider to enable the development teams to do operations. The transformation is a unique blend of changes in the domains of technology, people, culture, and process on both sides: development and operations.

We started publishing our experience with SRE on InfoQ in a data-driven decision-making article series[1] and later in a corresponding eMag.[2] The SRE article[3] from the series got attention, and I was approached to write a book on SRE transformation. The rest is history.

Publishing with Addison-Wesley is a privilege beyond imagination. While at university studying computer science, I read so many books from Addison-Wesley that I could identify them from a distance in the library. When I was offered the chance to publish a book with Addison-Wesley, I took it without much hesitation.

It is also a privilege to have some knowledge that might be worth publishing in a book in an industry that is very fast paced and where experience is not always valuable. At the same time, because of the pace of the industry and the bias for the new over the existing, it is a bit frightening that the knowledge I have is certainly not complete and will become obsolete quickly. More to the point, I seem to be one of the few people who was never affiliated with Google but has dared to write a book about SRE.

Further, lots of reading and gaining hands-on professional experience on my end led to a growing motivation to write. It is about giving back to the software engineering community at large where numerous authors of great books and talks shaped my thinking over the past decades.

Moreover, I consider it an entitlement in a world full of digital distractions to be able to work on a project that requires the highest levels of concentration. Writing a book certainly falls into this category. Writing this book taught me to stay away from digital distractions and develop an ability to concentrate quickly for longer time spans. It feels like my ability to concentrate is back to where it was before the era of connected devices.

My intention for the book is to support organizations that are starting an SRE adoption journey. The journey is a rewarding but difficult multiyear ride with lots of ups and downs. Adopting SRE means changing the culture, organization, responsibilities, practices, and technology around product operations. Product operations is what matters to users and customers. They only interact with the products in production. So, tending to production better is about directly improving the user and customer experience. How do you tend to production better to measurably improve the user and customer experience? How do you establish SRE as a means to getting there? How do you transform the organization toward SRE? This is what we will explore in the book.

The book is divided into three main parts. In Part I, "Foundations," you will establish a general understanding of SRE, its usefulness, and its place in the overall discipline of software operations. Additionally, I outline the challenge of SRE transformation in an organization new to the topic and explain how an organization's status quo can be assessed in terms of operations and readiness for SRE transformation.

---

1. Ukis, Vladyslav. 2021. "The InfoQ EMag: Effective Software Delivery with Data-Driven Decision Making." InfoQ, March 16, 2021. https://www.infoq.com/minibooks/data-driven-decision-making.

2. Ukis, "The InfoQ EMag."

3. Ukis, Vladyslav. 2020. "Data-Driven Decision Making – Product Operations with Site Reliability Engineering." InfoQ, March 25, 2020. https://www.infoq.com/articles/data-driven-decision-product-operations.

In Part II, "Running the Transformation," the transformation activities get rolling and unfold. For an SRE transformation to succeed, you must achieve proper organizational buy-in from the start. Here, I explain how to achieve this buy-in, initiate the transformation activities in the teams, and implement alerting, on-call rotations, and an appropriate incident response process in the organization. Accomplishing these tasks marks the establishment of the basic SRE foundations in the organization.

Part II continues with discussions about putting the advanced SRE foundations in place, including error budget policy and error budget–based decision–making. Following this, a suitable organizational structure for SRE is created. By the end of Part II, the organization has established the basic and advanced SRE foundations as well as an organizational structure for the long term.

In Part III, "Measuring and Sustaining the Transformation," I discuss how to measure the success of an SRE transformation and sustain the SRE movement. The book concludes with a look at the road ahead for SRE transformation beyond the established foundations.

Table P.1 shows the structural elements found throughout the book. They are embedded in the text and can be used as references on their own.

Table P.1 *Structural elements in the book*

Element	Description
Key Insight	A significant insight generated by the discussion in the book that is important to remember to be used in casual SRE conversations.
SRE Myth	A myth about SRE prevalent in the industry debunked in the book.
SRE Cheat Sheet	A reference of SRE topics to be looked up for a quick reminder.
From the Trenches	A story or insight based on hard-won lessons from the SRE transformation and practice. It is a description of what really worked at an organization in a particular context.

If you have any questions as you read through the book, feel free to reach out to me on LinkedIn.[4] I look forward to hearing from you!

Register your copy of *Establishing SRE Foundations* on the InformIT site for convenient access to updates and/or corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN (9780137424603) and click Submit. Look on the Registered Products tab for an Access Bonus Content link next to this product, and follow that link to access any available bonus materials. If you would like to be notified of exclusive offers on new editions and updates, please check the box to receive email from us.

4. Ukis, Vladyslav. n.d. "Dr. Vladyslav Ukis LinkedIn Profile Page." LinkedIn. https://www.linkedin.com/in/dr-vladyslav-ukis-5172ba32.

Acknowledgments

First I would like to thank my family. You are the emotional, intellectual, and spiritual foundation of my universe. My wife, Lina, is at the center. Being a UI/UX designer, she managed to listen carefully to me as I talked about a topic as technical as SRE, sometimes saying, "I got it, you do not need to go into more details right now!" It is Lina's enthusiasm, encouragement, and patience that made writing the book possible. Further, our children, Annika, age six, and Jonas, age two, are cheerfully around us. Jonas made it a habit to scroll through various black-cover and white-cover SRE books lying around at home. He might have learned something about reliability from them that could be applied at his age. It is our peace at home that made writing this book possible. Further, my parents; my brother's family; my mother's father, age 102; my uncles' families; my brother-in-law's family; my in-laws; the family of my wife's uncle and their children's families; other more distant relatives; as well as my friends all contribute to the firm family foundation I have that fueled this book project.

Second, as time is one of the most precious resources a person has, I would like to thank you, the reader, for taking the time to read this book. I hope the learnings from the book will shape your thinking about software operations in general and SRE in particular. My intention is to help make your SRE transformation as smooth and fast as possible. Get in touch; I long to find out how your SRE transformation is unfolding!

Siemens Healthineers is the center of gravity for my professional development. Specifically, the Siemens Healthineers teamplay digital health platform[1] has become a career-changing product suite and team. It provides an experimental environment necessary for introducing new ways of working. The new ways of working are tried out, and the ones that work well get adopted way beyond the teams that initially introduced them. This benefits the entire company in the end.

Specifically, a big thank-you goes to the teamplay leadership, Dr. Thomas Friese (previous executive) and Carsten Spies (current executive), for being open and supportive of my writing this book. With the book, SRE is going to be one of the most comprehensively documented processes at teamplay.

The QCon London conference became career defining as well. Literally all big organizational changes to improve technology at teamplay originated from talks, conversations, tracks, and meetings at QCon London. Both Continuous Delivery and SRE transformations originated there.

A big professional milestone for me was to meet Dave Farley at QCon London. He taught me and the entire team at teamplay the value, fundamentals, strategy, and tactics of Continuous Delivery. His way of thinking about Continuous Delivery is rooted in the scientific method. The scientific method is about answering questions using hypotheses that are put to the test with experiments. The application of the scientific method in the context of software development is why Continuous Delivery works. Interestingly, the application of the scientific method in the context of software operations is why

1. Siemens Healthineers. n.d. "teamplay Digital Health Platform." Accessed January 11, 2022. https://www.siemens-healthineers.com/digital-health-solutions/teamplay-digital-health-platform.

SRE works. My thanks go to Dave not only for Continuous Delivery but also for introducing me to Pearson.

On that note, a big thank-you goes to the book's executive editor, Haze Humbert, for the trust she put in my ability to write the book. She was very professional and a delight to work with from the moment of our first contact until the book's publication. Moreover, the manuscript reviewer, Niall Murphy, provided deep insights, uncovered flaws, and suggested improvements in a very timely fashion, which upon incorporation made the book so much better. Finally, the development editor, Mark Taber, the copy editor, Audrey Doyle, the production editor, Julie Nahil, the production project manager, Aswini Kumar, and many others made an incredibly effective and efficient team that turned the original manuscript into a high quality book. Thank you so much!

Specifically in the context of SRE, I am endlessly indebted to the teamplay operations engineer, Philipp Guendisch, for his enthusiasm around SRE as a discipline, dedication to driving it, and implementation of the SRE infrastructure at teamplay. It is his wit that made the SRE infrastructure reliable and a pleasure to use. Likewise, my thanks go to the many student interns supporting Philipp with the SRE infrastructure implementation.

My gratitude certainly goes to Google for coming up with the SRE concepts and turning them into a new computer science and software engineering discipline! Some Google insiders said that at the time the first Google SRE book was being written, there was not a single team in the Google SRE department doing things uniformly with other teams. So, compiling these different ways of working in a coherent set of SRE principles and practices was a tremendous task. This was, however, absolutely necessary to push SRE beyond Google. At some point the push reached me and, with that, the Siemens Healthineers teamplay digital health platform. The rest is history.

SRE conversations with Niall Murphy, one of the SRE pioneers at Google, and Steve Smith, one of the original operability thinkers at Equal Experts, shaped my thinking about many SRE aspects. Thank you for the time invested!

Interestingly, I had several people in my early development and early career who were particularly focused on establishing good processes in what they did. My father's father had worked as a chief technologist at a chemistry plant. He spent lots of time explaining to me the processes introduced at the plant to make the operations more efficient and effective over time. Although I did not understand the chemistry behind it, the outcomes of the process improvements were clear and exciting.

My friends during my school years fueled an initially rather modest interest in computer science. It is those collaborative conversations about our early programming attempts on calculators and PCs connected to tape recorders and TVs that ignited the sparks necessary to genuinely dig deep into the discipline.

My physics teacher in school, Vladimir Jakobi, taught the class to openly discuss the process of learning. Learning process sharing and improvement was one of the focus points in his physics lessons. It was unusual but had very positive effects on the students' learning outcomes. It taught me early on that the process of doing a thing right is as important as doing the right thing. Early in my professional career, Karlheinz Dorn at Siemens Healthineers taught me the value of a disciplined process in the context of software architecture.

Moreover, I am very thankful to Prof. Dr. Stefan Jablonski for supervising my bachelor thesis at the University of Erlangen-Nuremberg in Germany, as well as Gerold Herold for supervising my masters thesis at Siemens Healthineers. These signficant projects gave me unique opportunities to

grow professionally in technical, interpersonal, and organizational dimensions. The associated thesis write-ups showed me the value and impact of clear technical writing.

Further, big thanks go to Kung-Kiu Lau, my PhD supervisor at the University of Manchester in the UK. It is his endless patience that honed my writing skills. I remember numerous meetings in his office discussing our joint research papers. Me calling out, "How can I explain this to somebody who does not know anything about computer science?" was a rather frequent question in these meetings. Undaunted, Kung-Kiu kept at it until our research papers could be understood by people without a background in what we were writing about. As a result, my writing skills and speed improved over time too.

Writing this book certainly required a strict writing routine to be introduced into a busy professional and family life. Being mentally prepared for the need of such a routine to be put in place and stuck to for a long time simplified the decision to write the book and follow through on all aspects of publishing.

There is an interesting quote about writing by E. L. Doctorow: "Writing is like driving at night in the fog. You can only see as far as your headlights, but you can make the whole trip that way."[2] I can relate well to the quote. It is amazing how much information the brain contains on just a single topic in a condensed and foggy structure, which gets uncompressed on hundreds of pages in a form that can be learned from by others.

Before I embarked on my PhD studies, many people said that doing research would be a unique opportunity to focus on a single topic which would not present itself in my future professional life. I guess that was not quite right. Writing this book certainly allowed me to focus on the subject of software operations as much as I focused on software architecture back in the days of my postgraduate studies.

An anecdote is that the original manuscript of the book was written in Google Docs. As I was writing about SRE, I was thinking about how the Google Docs SLOs might get broken while I was writing. Knowing the level of rigor applied to the SRE process by Google contributed to my peace of mind that even if Google Docs SLOs get broken, the services will be brought back within the SLOs rather soon. Writing about SRE using a word processor operated using SRE by the company that invented and practiced SRE might be one of the best representations of "eating your own dog food."

Finally, this book should serve as an inspiration to the world of writing to my daughter, Annika, who started school in 2021, the year the manuscript of this book was finished. Likewise, it should inspire my son, Jonas, who started learning to read letters the same year, to continue by combining them into syllables, words, sentences, paragraphs, stories, and, finally, books. I enjoyed writing the book and, throughout the process, realized that I might wish to write another one in the future.

These people and organizations influenced me to a great degree. I am very appreciative of being in such an innovative professional and caring family environment. Simply said, you all put me where I am today. Thank you!

2. Doctorow, E. L. n.d. "A Quote from Writers At Work." Accessed January 8, 2022. https://www.goodreads.com/quotes/53414-writing-is-like-driving-at-night-in-the-fog-you.

Coming originally from Ukraine, I am compelled to extend my deepest sympathies for the innocent civilians who remain or have been forced to flee Ukraine because of the current war. I stand in solidarity with Ukrainians during this humanitarian crisis. I join the UN General Assembly resolution demanding an end to this Russian offensive in Ukraine.

Vladyslav Ukis
April 2022

About the Author

Dr. Vladyslav Ukis is Head of R&D for the Siemens Healthineers teamplay digital health platform and reliability lead for all Siemens Healthineers Digital Health products. Previously, as software development lead, he drove Continuous Delivery, SRE, and DevRel transformation, helping this large distributed development organization evolve architecture, deployment, testing, operations, and culture to implement these new processes at scale.

Dr. Ukis earned a degree in computer science from the University of Erlangen–Nuremberg, Germany, and later from the University of Manchester, UK. During his career, he has been working on software architecture, enterprise architecture, innovation management, private and public cloud computing, team management, engineering management, portfolio management, partner management, and digital transformation at large.

This page intentionally left blank

Chapter 2

The Challenge

In Section 1.2, *Alignment Using SRE*, I presented an example of how a product delivery organization works without alignment on operational concerns. The example showed that without alignment, operational concerns are addressed only once production issues occur. This is done using ad hoc urgent meetings involving product operations, product development, and product management. The example is representative and can be generalized to better understand the challenge of SRE transformation.

A product delivery organization unaligned on operational concerns does not weave aspects of operations consistently and evenly throughout the product creation life cycle. Operational concerns are seen by most, as the name suggests, with production operations. Because product operations is the last part in the chain of product management, product development, and product operations, people think about operational concerns as the last thing on their to-do list. This is not a product-centric way of thinking. Users touch the product in production. Therefore, that touch point needs to be centric with all activities in the product creation life cycle. Indeed, product operations needs to be elevated and treated on par with user research, user story mapping, user experience design, architecture, and development.

The consequence of not thinking about production throughout the product creation life cycle can be illustrated using an example from the grocery industry. Imagine that a grocery store chain has a wide variety of products displayed in beautifully designed stores throughout the country, but neglects the checkout counters at the point of sale. The entire supply chain is working flawlessly, but issues arise at the checkout where the customers are trying to purchase their groceries: for example, they might not be able to pay for their groceries quickly, and the checkout queues might be getting longer. The checkout staff might not be able to resolve the issues themselves. The point-of-sale devices are supported by the operations team, which receives an enormous number of support requests. It turns out that the issues are with the software on the devices; the support team cannot resolve the software issues themselves.

While the crisis is unfolding, the developers are happily working on new features for the point-of-sale devices. The product owners are happily specifying additional new features to be handed over to the developers after they finish the current work. The operations engineers are reaching out to the developers, who are not sure whether to prioritize the requests by the operations engineers or the features in development. The developers reach out to the product owners for a prioritization decision. Finally, the operations engineers, developers, and product owners swarm over the problem and decide to fix the product issues with the highest priority.

2.1 Misalignment

Figure 2.1 illustrates the preceding example of how a product delivery organization misaligned on operational concerns works.

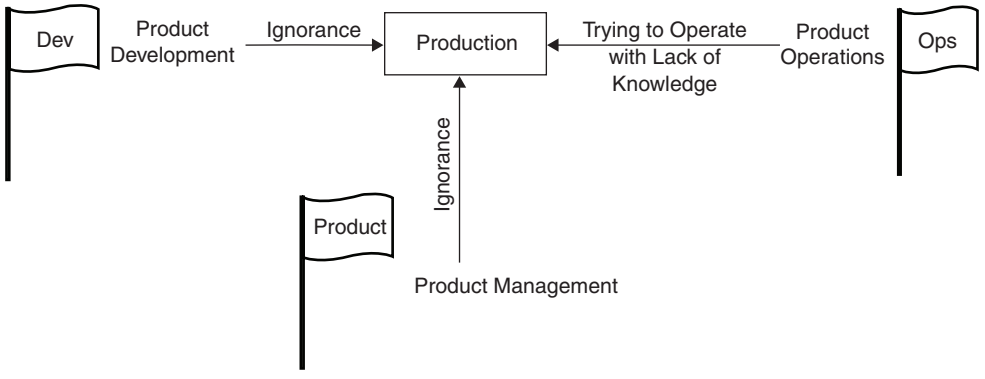


Figure 2.1 *Product delivery organization misaligned on operational concerns*

The left-hand side of the figure shows how product development is working on the feature backlog prioritized by product management. By and large, product development ignores what is going on in production. There is no ongoing visibility into how the system is performing in production. Nor have they set up any alerts to be notified about abnormal situations. Product development's focus is entirely on new feature development. Product operations is not part of their backlog.

Product operations is shown on the right-hand side of the figure. The product operations team is trying really hard to operate the product in production. However, they lack insider knowledge about the product in order to be able to operate it properly. This insider knowledge is with product development. Furthermore, this knowledge is changing quite rapidly with new releases being deployed to production on a frequent basis. Lacking insider knowledge about the product in operation, the operations team sets up alerts on technical resources that are visible outside. These are parameters such as memory consumption, CPU utilization, queue fill levels, disk storage fill levels, and network monitoring, among others. The parameters' thresholds are alerted upon. Once the alerts arrive, the operations team tries to understand whether there is anything wrong with the system. Often, they have to consult the product development team to analyze potential issues. The issue backlog is growing, which frustrates the product operations team. They do not understand product development's attitude toward solving issues in production. If production is where the customers use the product, how on earth can it be less important than anything else?

This frustration reflects a core issue in product delivery organizations that do not excel at operations. In such organizations, being product and user centric means different things to different parties. From a product operations point of view, it means production issues are tackled with the highest priority. From a product development point of view, it means features requested by product owners are developed as quickly as possible. From a product management point of view, it means user stories requested by customers are turned into features in production as quickly as possible. This fundamental misalignment of what it means to be product and user centric when approaching product creation is one of the core reasons for difficulties in operating

the product in production to the customer's satisfaction. This is where SRE contributes greatly to aligning the parties in the product delivery organization.

The product management discipline is depicted toward the bottom of Figure 2.1. Product management is very far away from production thinking. In their view, it is a job for product operations to resolve. The product management team are busy talking to executives, stakeholders, customers, partners, and users, trying to figure out where the product fits in the market, identify missed user journeys, pinpoint ways to optimize workflows, and so on. Product management maintains a backlog of features to implement. Although the backlog is prioritized, as mentioned earlier, it doesn't consider product operations requirements. The product management team expects product development to develop and product operations to operate. This is what the names of the departments suggest, do they not?

In essence, the three parties in the product delivery organization operate under three different flags, as depicted in the figure. Product operations is proud to run under the Ops flag; they man production. Product development runs under the Dev flag; they are proud developers of new features. Product management runs under the Product flag; they are all about the product and shape it in a fundamental way: What is it? Who are the customers? What is the competition? What is the product's competitive advantage? What are the most important user journeys? What are the features? What is on the backlog?

It turns out that in a setup like this, no one really owns production operations. Who is it, indeed? Is it product operations? Not really, because they lack the knowledge necessary to truly own production operations. There is no proper continuous knowledge transfer from product development and product management toward product operations, and vice versa.

Is it product development? Certainly not. Their focus is the feature backlog. The feature backlog is void of product operations. Shipping occasional necessary production hotfixes after escalations from product operations is not what owning production operations actually means.

Is it product management? For sure it is not. Their focus is the definition of the product. Their expectation is that product development implements the product and product operations operates it in production. Despite the word *owner* in their title, the product owners do not own the product all the way to and including production.

In this context, it is no wonder that it is precisely in production that the product ends up being neglected. Where there is no ownership, there is no commitment. It would require commitment from all the parties in the product delivery organization to contribute to product operations in production. But how? Who would need to commit to what to establish a meaningful partial ownership of product operations? Would the ownership of product operations be a collective ownership, then? Let us explore these questions in detail.

2.2 Collective Ownership

According to Wikipedia, "collective ownership is the ownership of means of production by all members of a group for the benefit of all its members."[1] The definition shows that everyone needs to benefit from the ownership. In the context of product operations, it means that if collective ownership is to be established in a product delivery organization, the ownership needs to

1. Wikipedia. 2021. "Collective ownership." https://en.wikipedia.org/wiki/Collective_ownership.

benefit all the parties involved. Specifically, if collective ownership of production operations is to be established among product operations, product development, and product management, each party needs to benefit from it.

This is an interesting point to delve into. In the product operations team's view, they own product operations. However, they encounter great difficulties engaging the product development and product management teams in their operations activities. Therefore, product operations will welcome it if the product development and product management teams take partial ownership of production operations.

In the product development team's view, they are feature developers. Shipping new features to production quickly is at the core of their activities. What kind of benefit would they gain if they were to own production operations in some partial manner? What would it look like? Would the backlog contain operational user stories? This is not really feasible, as the operations work is not predictable compared to feature work that can be planned using a feature backlog. What would be beneficial is if the partial ownership of production operations led to insights that would lead to an improved development process augmented by a full operational context. In turn, it would be beneficial if the improved development process led to the reduction of production issues that interrupt feature work.

In the product management team's view, they define the product. The features should be developed by the product development team and operated by the product operations team. What would be the benefit for product management to own production operations in a partial way? To answer this question, customer escalations need to be looked at. Product management particularly dislike customer escalations. Customer escalations disrupt their work, require immediate focus, take a lot of time to justify the product to various stakeholders despite customer dissatisfaction, and chip away the stakeholders' trust. Diminishing stakeholder trust might lead to budget reductions for the product. This is a difficult situation every product owner works to avoid. To be sure, every customer escalation is about an issue in production. So, if the partial ownership of production operations would lead to a reduction of customer escalations, it would be a great and welcomed benefit for product management.

Table 2.1 shows the benefits each party would see in a product delivery organization if a collective ownership of product operations were established.

Table 2.1 *Benefits of Collective Ownership of Production Operations*

	Discipline	Benefit
Collective ownership of production operations	Product operations	Appropriate engagement of product development and product management in operations activities as needed. No more chasing product development and product management on every production issue to decide how to proceed.
	Product development	Appropriate insight in product operations to get to an improved feature development process augmented by the full operational context. Feature development performed with the full context of what is necessary to make the features technically successful in production leads to a reduction of customer escalations. This leads to more uninterrupted time for working on new features.
	Product management	Reduction of customer escalations and time investment to handle them.

Having clarified the benefits a collective ownership of production operations may bring to product operations, product development, and product management, the next question to explore is how to get the benefits. An associated question would be the cost of getting the benefits for each party involved. In other words, in the context of an SRE transformation, how do you implement collective ownership of production operations using SRE with a positive cost–benefit ratio?

2.3 Ownership Using SRE

What does it mean to have partial ownership of production operations using SRE? This question needs to be answered specifically for each party in the product delivery organization.

2.3.1 Product Development

In product development, the benefits of partially owning production operations are rooted in the insights of how the system behaves in production under real user, data, and infrastructure load. The most effective way to continuously learn about a system in production is to observe it in production. This is done using on-call rotations. Traditionally, product operations would go on call for services in production. This way, production insights do not go directly to product development. It follows that product development needs to get involved in on-call rotations for their services in production. Each development team owns some services. For exactly those services, the respective developers would need to be involved in on-call rotations to gain insights from operating the services under real conditions. These insights lead to the following improvements in product development and operations.

- Developers with product implementation knowledge conduct product failure investigations.
- The number of steps in the chain between a production issue occurring and a person with the best knowledge to fix it can be exactly one. The issue can go directly to the developer who implemented the service and can fix it the fastest, provided the alerting is targeted well and there is an agreement with the product owner to fix production issues immediately. The developer can take the learnings from the failure itself, the failure analysis, and the fix into the new-feature development process, supporting infrastructure and debugging tools. This should lead to the product being more operable in the future with less time required to operate it. In turn, it should lead to more time for feature development.
- Developers get to experience the quality of the product in the real world by testing it at production sites. Internal testing is rarely as intensive as the strain a system undergoes in production. Seeing real-world scenarios informs the development of automated test suites and contributes greatly to closing the gap between the internal testing and production scenarios. Thus, confidence is increased in deploying the product to production once the test results from the internal test suites are green. This should lead to fewer failures in production due to scenarios that were untested internally. In turn, it should require less time to fix production issues, leading to more time for feature development.

- Developers gain the knowledge necessary to operate and troubleshoot the product. This informs the development process, among other things, leading to better tools for operations. In turn, it leads to less time spent on troubleshooting production issues, which frees up more time for feature development.
- Developers use the knowledge from product operations in the development of new features. For example, scalability and performance requirements can be learned, and this can often lead to architecture changes. Although making such changes requires a significant amount of work, it is necessary to implement resilience in accordance with the load profiles seen in production. Only then can the system's operational burden (also known as *toil*) reduce, freeing time for more feature development.
- Developers gain a better understanding of the kind of testing and tooling necessary to deliver a product that works well. Test scenarios, test levels, test runs, and test environments need to be designed in such a way that all the testing activities combined address important scenarios the system encounters in production. To be sure, production itself can be one of the test environments with tests running there 24/7. Gaining insight into product operations in production can greatly inform the entire test management process. This should lead to test suites and test runs being more focused on the scenarios taking place in production, which can reduce the amount of time spent on tests that are not effective in catching bugs encountered in production, as well as the rework and maintenance of such tests. Streamlining test management may lead to more time available for feature development.
- Developers have the incentive to implement reliability features and tools for a great product operations experience. This is because if the developers go on call, they actually want to spend as little time as possible dealing with production issues. In this context, they have full control of the situation. It is in their power to implement the product with production operations in mind. Doing this leads to spending as little time as possible on production issues and maximizing the time spent on feature development. This benefits customers and product management alike. Customers also do not want to deal with product failures in production. Rather, they want existing features to work in production and new features added to the product quickly. The product management team, driven by customer requests, wants product development to work on new features.
- Developers with experience in product operations are more highly valued in the industry. Going on call directly contributes to learning the skills necessary to command higher wages in the marketplace.

The idea of going on call for the developers gives rise to a plethora of questions, such as the following.

- Do the developers always need to go on call for their services? No.
- Could the developers go on call only during business hours? Yes.
- Can the on-call responsibility be shared with product operations? Yes.

- What is the best setup for a given organization? It depends.
- Would a development team setup need to be adapted to enable on call? Yes.
- Can developers in a team perform the on-call duties on rotation? Yes.
- Can focused feature development still be done despite going on call? Yes.
- How do you achieve it? It depends.
- Can developers stay developers if they go on call? Yes. They will become better developers. Their skills will be more highly valued in the job market.

These and other related questions will be explored in the book in due course. It is not necessary to answer them in-depth here. For now, I will only outline the scope of the challenge posed by the SRE transformation. What is important to understand at this point is that product development needs to go on call to one extent or another depending on the organizational setup chosen for the organization's SRE implementation.

Without developers going on call to some extent, the benefits of collective ownership of production operations cannot be realized by product development. Feature development is difficult to improve from an operational standpoint without a live feedback loop between the production and development teams. An outage profile in production cannot be sustainably influenced if the feature development team is not well informed using the live feedback loop from production experienced by those who implement the features. In other words, without developers going on call to some extent, things in product development remain the same by and large as far as operational concerns go.

Key Insight: Developers must go on call for some percentage of their time. This can range from very little time to nearly full time.

This key insight is illustrated on the left-hand side of Figure 2.2.

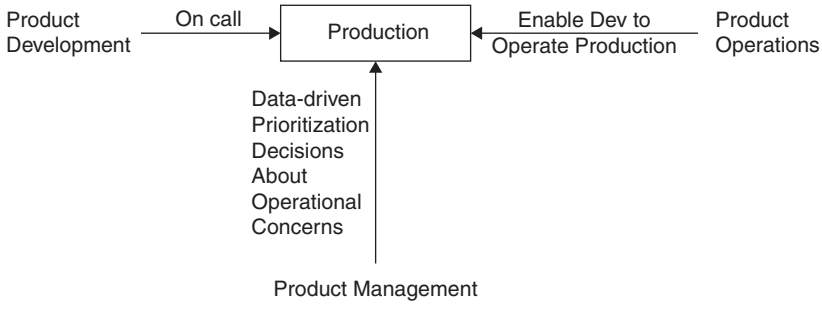


Figure 2.2 *Collective ownership of product operations using SRE*

2.3.2 Product Operations

The product operations discipline is depicted on the right-hand side of Figure 2.2. With developers going on call, the product operations team would need to provide support to enable the developers to do operations.

What kind of support would the developers need? They may never have done operations before, in which case this will be unfamiliar turf for them. Are there trainings for this? Does the operations team provide some onboarding? What does "good" look like in product operations? Is there any documentation available? These are the questions that come to mind for developers when confronted with going on call for the first time.

The entire body of knowledge about product operations is with the operations team. But what kind of knowledge is that? Mostly it is about taking the product as a black box, putting it into the production environment, activating monitoring of IT resources, and alerting on some threshold violations. Developers can learn and understand this. With their insider knowledge of the product, they will also be able to find many more scenarios that can be monitored and alerted upon. The developers' knowledge about the architecture, implementation, configuration, and deployment of the product is an invaluable resource for improving monitoring of the product in production. But how can they utilize that knowledge to improve product operations? How can they bridge the gap between development and operations as suggested by the term *DevOps*?

Let us look more closely at what the developers know. They know how specific routines that contribute to the fulfillment of user requests are implemented. They know the paths the user requests go all the way from the user interface to the deepest service in the service network, and from there to the infrastructure. If the product exposes APIs to customers, the developers also know the paths the API requests take from the API gateway through the network of services all the way down to the infrastructure. Moreover, the developers know which services they implemented versus those implemented by the company and third parties. They know which third-party services are difficult to integrate with, where the domain model of the third-party services is overly complicated and cluttered, where the third-party services are slow occasionally, and where there is simply sporadic behavior that can be explained. The developers also know all this for the internal services of the company, which are the services they depend upon.

Their knowledge does not stop there. The developers and architects know the strengths and weaknesses of their architecture. They know where the architecture limitations lead to performance and scalability issues. They know the circumstances where the performance and scalability issues are likely to exhibit and probably impact the customer. They know the architectural debt in the system and which part of it is planned to be paid off in the near future. They know of any major architectural refactorings that must take place, which are not planned due to the size of the effort involved.

The developers' knowledge goes much further. They know about the infrastructure limitations the product is running on. They know how each service can impact the others; for example, they know what will happen if a particular service in the service network eats up the lion's share of memory in a given area of the infrastructure. They might know some parameters of the container clusters the services are running in and anticipate issues that might occur based on the changing data and user load profiles.

There is yet more to the developers' knowledge. They may know the way the services are deployed: Which infrastructure parameters are set by the deployment infrastructure, and which ones are set in the service at the deployment time, startup time, or runtime. They know which services are deployed independently, which ones use a shared deployment pipeline, and which ones are deployed manually for the time being. They may know the tests running on the deployment pipelines, the quality of those tests, and whether the test results can be trusted. They may know the test management process for a service, the test levels available, the test infrastructure, and any test gaps that exist.

Additionally, the developers might be aware of security implications in the architecture and implementation. Which security vulnerabilities are taken care of? Which are mitigated? Which are known but are not currently taken care of ? Which bugs from penetration testing were not yet fixed?

Finally, the developers know the most painful product areas from a development point of view. What area is the most difficult to integrate with? To test? To speed up? To debug?

This amount of knowledge is staggering to the operations engineers. How do they take all this knowledge from the developers and apply it to product operations? Can it be done with some tool support? What kind of role would automation play here? Does it all sit between the ears of the software developers and cannot be easily repurposed to improve product operations? How can it support the developers effectively?

In other words, the developers know the car engine from the inside. But how do you help them use that knowledge to improve how the car operates?

To approach these questions, we need to turn our attention to how developers make known to the outside world what is going on with the system on the inside. This is done using logging. During development, developers decide what to log and under what circumstances. This way, once the product runs in production, log entries are generated that contain logging information. The log entries stored in, for example, log files or other storage systems can then be analyzed to understand what was going on in the system at runtime. This is the basic process of how developers make known to the outside world what is going on inside a system at runtime. The process is sophisticatedly supported by tools providing all sorts of runtime instrumentation out of the box. That is, the developers' knowledge about the product can be encoded in logs that can be analyzed outside the system.

The next question to ask is what should be logged to improve product operations? Let us imagine, these questions would be answered.

Once that question is answered, we would consider how to log relevant information in a uniform way. What should be the log format? Which log format would lend itself to automated log processing? Would several log formats be required for different operational aspects; for example, one log format for calculating service availability and another for calculating service latency? What about asynchronous operations—how do you log those? Where do you store the logs? Should the logs be stored in regional data centers or centrally? How long should the logs be stored? Let us imagine, also these questions would be answered.

With the answers to these questions, we would next consider how to detect abnormal situations. What should be considered broken availability? What should be considered broken latency? What should be considered insufficient throughput? Which aspects beyond availability, latency, and throughput are important to consider? Let us imagine, these questions would be answered too.

Next, we would want to know how to alert in abnormal situations. Should alerts be generated as soon as the abnormal situation has been detected, or a bit later? Should the alerts be sampled? How do you avoid alert fatigue, in which those who receive the alerts become overwhelmed with too many alerts and stop reacting to them? How do you strike a good balance between alerting people so often that it causes alert fatigue and so rarely that it causes incidents to go unnoticed? What kind of information needs to be included in the alert? Even if these questions would also be answered, there would be more.

The next questions would be about whom to alert—specifically, which developers receive the alert? How do you alert developers in such a way that they do not get distracted from their feature development work? How do you alert developers in such a way that they will actually react to the alerts, provided the alerting does not lead to alert fatigue? Can any developer in general be alerted? What kind of knowledge would a developer need to have to be able to react to alerts within a reasonable time frame and with reasonable effort?

The list of questions can go on. What it shows is that a comprehensive framework that would enable developers to conduct product operations is required. But what is a framework? According to Wikipedia, "a software framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software."[2] So, what is needed in the context of an operational framework is some generic functionality that can selectively be changed. In the context of SRE, a framework like that can be referred to as SRE infrastructure. It needs to provide generic functionality supporting the use cases exemplarily outlined previously, implemented within an SRE context. The generic functionality needs to be selectively changeable to adapt the infrastructure to a specific use within the overall set of SRE activities.

> **Key Insight:** Operations engineers need to provide frameworks to enable developers to do service operations. In an SRE context, such a framework can be referred to as SRE infrastructure.

At the time of this writing, some off-the-shelf tool support for the SRE infrastructure exists, but it is not comprehensive enough to eliminate the need for custom development of missing pieces. Therefore, in all likelihood, building an SRE infrastructure is going to require some custom software development combined with ready-to-use off-the-shelf tools. This means product operations would need to learn to do software development.

The challenge for product operations is lack of experience providing frameworks that enable others to do operations work. The product operations has always conducted operations work in a hands-on manner using existing tools. What is required from product operations now is the enablement of product development to perform service operations. The enablement is done using the envisioned SRE infrastructure. The SRE infrastructure needs to be built using first-class software development techniques.

This is in line with SRE and the words of Benjamin Treynor Sloss: "SRE is what happens when you ask a software engineer to design an operations team." Following this, it should be no surprise that enabling the product development team to do operations work requires the

2. Wikipedia. 2022. "Software framework." https://en.wikipedia.org/wiki/Software_framework.

software development team to build a suitable SRE infrastructure. Building frameworks is common in software development. Using frameworks is familiar to software developers. Neither of these will be familiar turf for operations engineers from the product operations discipline.

The following now unfolds as a challenge in SRE transformation.

- Software developers need to learn how to do product operations work by going on call.
- Operations engineers need to learn how to enable software developers to do operations work by developing the SRE infrastructure as a framework.

This is illustrated in Figure 2.3. The two arrows resemble the moves from fencing. It might sound ironic, but this is exactly what needs to happen during SRE transformation.

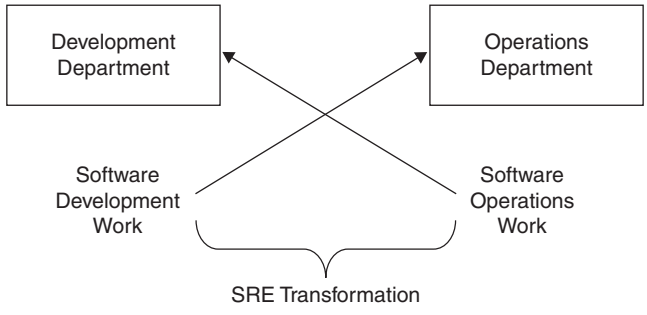


Figure 2.3 *Key SRE transformation challenge*

Neither of those arrows is easy to achieve. However, as evidenced by the growing number of software delivery organizations around the world, it is entirely possible, and will be explored at length in this book.

Figure 2.3 shows what it truly means and takes to implement DevOps. It is about developers doing operations work and operations engineers doing development work. It goes to the heart of both long-standing disciplines, product development and product operations, and shakes their fundamental responsibilities. Truly implementing DevOps takes far more than just achieving good collaboration between product development and product operations.

The difficulties are especially great in traditional software delivery organizations. A development organization that has never done operations and an operations organization that has never enabled others to do operations lack the very foundations on which SRE can be established. Developers do not understand why they should be doing operations. Operations engineers do not provide frameworks to enable developers to do operations. Managers do not promote the endeavor, let alone fund it.

Despite these difficulties, it is well worth the effort to embark on an SRE transformation. The kind of DevOps implementation that can be achieved using SRE is where developers maximize their feature development time while having evidence that the product works well for customers in production. Without SRE, developers maximize their feature development time, ignoring production.

Further, in a DevOps implementation using SRE, operations engineers scale well by providing the SRE infrastructure to the developers, which enables them to do production operations.

Without SRE, the operations engineers are the bottlenecks. They do production operations purely by themselves, regardless of the product quality and insider knowledge available about the product.

2.3.3 Product Management

Having clarified what collective ownership of production operations means for product development and product operations, it is time for such a clarification in the context of product management. What does the product management team need to do to partially own product operations?

Traditionally, product management is pretty far away from product operations. As discussed in Section 2.2, *Collective Ownership*, product management's benefit in getting involved in production operations is to reduce customer escalations. How on earth can product management reduce customer escalations if everything the customers escalate about is a technically broken product? Product owners are not technical experts. They neither implemented nor deployed the product.

To approach this, let us explore what leads to customer escalations. Before a customer gets to the point of picking up the phone and calling customer support to complain, a series of events take place. The customer works with the product and notices something annoying. It might be a sluggish display of data; an inconvenient way of accomplishing a task in too many back-and-forth steps; an action taken, like a button click, that does not result in the action actually performed; or a downright crash with accompanying data loss. Whatever the reason, it is directly linked to the customer having lost so much time or money that they call customer support to release their anger and get help.

Now, could the technical experts—namely, the product development or product operations team—have noticed anything wrong with the product and fixed it earlier? Are product development and product operations set up for such incident detection and resolution? Again, this is technical, so what does it have to do with product management?

Let us dive deeper. Imagine that product development and product operations want to set up incident detection and resolution to detect and fix abnormal situations before customers escalate. How would they go about doing this?

As you saw in Section 2.3.1, *Product Development*, the developers have an enormous amount of knowledge about all sorts of technical aspects regarding the product. The operations engineers have vast experience with customer escalations. They remember a lot of past escalations by heart. They can cluster them. They know by means of anticipation the weak areas of the product that are going to be escalated about soon because product development has not started fixing them. Overall, this is a good mix of knowledge that is brought to the table by product development and product operations. The product development team brings knowledge of technical implementation while the product operations team brings knowledge of the actual issues from production. Taken together, this knowledge can be used to create an incident detection and resolution process rooted in technical implementation and past customer escalations. This is great. It would be a huge leap from ad hoc, unsystematic incident response. It would reduce customer escalations.

The goal, however, is to aim higher. The goal is to create an incident response and resolution process that for every existing and new feature would detect abnormal situations early enough

for product development to fix and then to deploy the fixes before customers escalate. This would be a real benefit to product management. To emphasize, the process should work for every existing and new feature, not just for features known to product operations based on the experience of past customer escalations. Also, to emphasize, the developers would allocate their time in such a way that they fix detected issues and deploy the fixes to production before the customers get angry enough to escalate. This means the developers would not just work on the feature backlog prioritized by the product owners. The other prioritization driver would be the product reliability issues detected by the incident response process.

With that, the contribution of product management to the collective ownership of product operations is starting to emerge.

1. Product owners would need to contribute user journey knowledge to the incident detection process. Impaired and broken user journeys should be at the core of incident detection. Which user journeys are the most important ones to detect incidents with? What are the most important steps within a given user journey that must work for the user journey to still make sense? Conversely, which steps of a user journey could fail, and how badly, without rendering the entire user journey broken? Overall, the incident detection process is as good as the defined incidents it can detect. To define detectable incidents well, the user journey knowledge of the product owners, the implementation knowledge of the developers, and the operations knowledge of the operations engineers need to be combined.
2. Product owners would need to understand and agree to the importance of setting up a backlog management procedure in which developers can flexibly allocate time to fix production issues as they are detected by the incident detection process. Traditionally, the product owners prioritize the backlog of user stories, and they want developers to focus on the backlog. To reduce customer escalations, the product owners would want the developers to take immediate action on the issues reported by incident detection.

This now makes sense to the product owners. They were part of and shaped the incident detection definition. They know what the incident detection is going to detect. It is going to detect real broken user journeys and not merely some technical deviations. Now it is easier for the product owners to accept the engineering time being spent on incident resolution. Why? Because spending that time directly contributes to the reduction of customer escalations. If developers do not fix the incidents in production within a reasonable time frame, the customers will still escalate despite the right incidents being detected early enough.

That is, to reduce customer escalations, the following criteria need to be fulfilled.

- The incident detection detects broken and impaired user journeys as defined together by the operations engineers, developers, and product owners.
- The developers prioritize fixing broken and impaired user journeys as they are detected without having to negotiate with product owners every time about the engineering time allocation.
- The developers fix the broken and impaired user journeys in production within a specified time frame before customers get angry and frustrated enough to escalate.

This process is shown in Figure 2.4.

The top left of Figure 2.4 shows the incident detection definition process. It takes as input the implementation knowledge by developers, the operations knowledge by operations engineers, and the user journey knowledge by product owners. The outcome of the incident detection definition process is an understanding of the incidents to detect in production. It is about detection of unhealthy patterns in

- The user journeys from an operational criticality perspective
- The critical service dependencies fulfilling the user journeys
- The critical infrastructure components and their scaling, fulfilling the user journeys

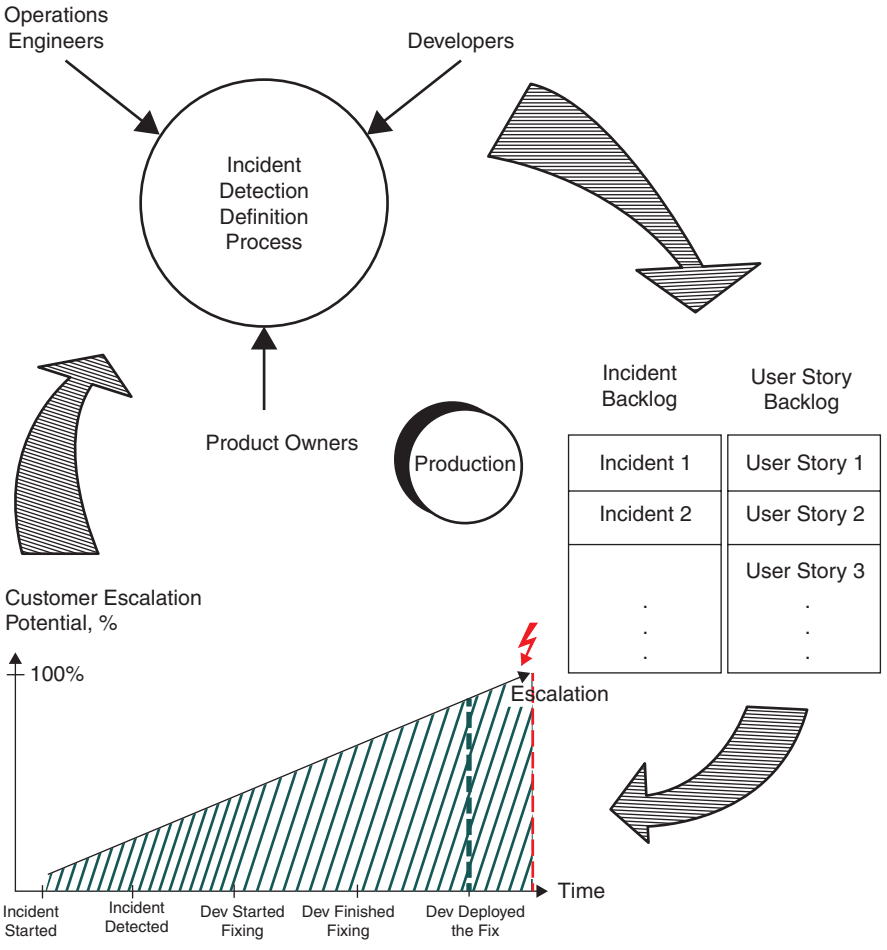


Figure 2.4 *Process for reducing customer escalations*

With the definition of what to look for in production to detect incidents, incident detection can be switched on. The production environment in the middle of Figure 2.4 can now run under monitoring for fulfillment of real user journeys, instead of having monitoring for fulfillment of technical parameters. Monitoring for fulfillment of real user journeys is more targeted to reduce customer escalations, which is product management's benefit in terms of getting involved in product operations.

Once the incident detection process has detected some incidents, it will put them into a backlog. This is shown on the right-hand side of Figure 2.4. The incident backlog exists side by side with the user story backlog. The user story backlog is prioritized by the product owner. The incident backlog also needs prioritization. This prioritization needs to be done as a just-in-time process while the incidents get detected. It also needs to be done very quickly. Lengthy negotiations between the operations engineers, developers, and product owners cannot take place to efficiently prioritize the incident backlog. This means prior agreements among the three parties need to be reached. A good place for timely incident prioritization agreements is the incident detection definition process itself. As part of the process, not only are the incidents defined but also their relative priority can be agreed. These agreements should enable all the people on call, and especially the developers, to make autonomous incident prioritization decisions for a majority of the incidents.

Because the incidents from the incident backlog need to be worked on in a just-in-time manner, the developers working on the incidents cannot work on the user stories from the user story backlog at the same time. Also, just-in-time switching between the incident backlog and the user story backlog leads to a great context switching overhead. Not only is this inefficient, it also places a significant mental burden on the developers. To counter this, there are strategies to setting up development teams in such a way that the ongoing on-call work from the incident backlog and the focused user story work from the user story backlog are well balanced. These strategies will be explored later.

The bottom of Figure 2.4 shows the incident processing timeline. It begins with an incident, shown on the far left. At the time of the incident, the potential for customer escalation is very low. It grows to 100% over time, which is the point to avoid. The goal is to fix the incident before the customer gets angry and frustrated enough to call customer support with an escalation.

After the incident started, it can be detected by incident detection. The next step on the timeline is the point in time when the developer starts working on the fix. Once the issue has been fixed, it needs to be deployed to production. Once it is deployed, the fix needs to be monitored to ensure that the incident has truly been resolved. The goal is to perform the fix deployment and associated monitoring, confirming the incident resolution before the red line of customer escalation.

The incidents that have started may go unnoticed, or be noticed too late or too early by incident detection. Conversely, the incidents may represent false positives. This happens when an incident is reported that does not lead to deterioration in the user experience. All these cases need to serve as input for adjusting the incident detection definition. This is an important part of the overall process. It enables the incident definition adjustments to be done regularly based on the real feedback loop from production. The feedback loop is data driven. This enables the three parties—product operations, product development, and product management—to decide on the incident definitions in an opinion-neutral, data-driven way.

In the context of SRE, such an incident detection and response process is set up using specific mechanisms and terms, such as service level indicators (SLIs), service level objectives (SLOs), and error budget policies. Soon, an exploration of these concepts will begin. Before this exploration, let us summarize the benefits and costs of the collective production operations ownership using SRE.

2.3.4 Benefits and Costs

The analysis in the previous chapter showed what it would take for product operations, product development, and product management to truly work together as a team using SRE methodology. It showed the deep integration among the three parties necessary to implement DevOps using SRE. It takes much more than only a good collaboration among the three parties. Table 2.2 juxtaposes the benefits and costs.

Table 2.2 *Benefits and Costs of Collective Ownership of Production Operations Using SRE*

	Discipline	Benefit	Cost
Collective ownership of production operations using SRE	Product operations	Appropriate engagement of product development and product management in operations activities as needed. No more chasing product development and product management on every production issue to decide how to proceed.	Enabling others to do operations by implementing SRE infrastructure as a framework.
	Product development	Appropriate insight in production operations to get to an improved feature development process augmented by the full operational context. Feature development performed with the full context of what is necessary to make the features technically successful in production leads to a reduction in customer escalations. This leads to more uninterrupted time for working on new features. Additionally, there is a developer skill upgrade valued by the job market.	Doing product operations by being on call during defined times.
	Product management	Reduction of customer escalations and time investment to handle them. Ad hoc involvement in numerous production issues is also reduced, and there is an added ability to make decisions in a data-driven manner about engineering capacity allocation to features versus operational concerns.	Involvement in the incident detection definitions and data-driven prioritization decision-making based on production data.

Now that the benefits and costs of the common ownership of production operations using SRE are clear, let us take a look at the overall picture of what SRE is trying to achieve (Figure 2.5).

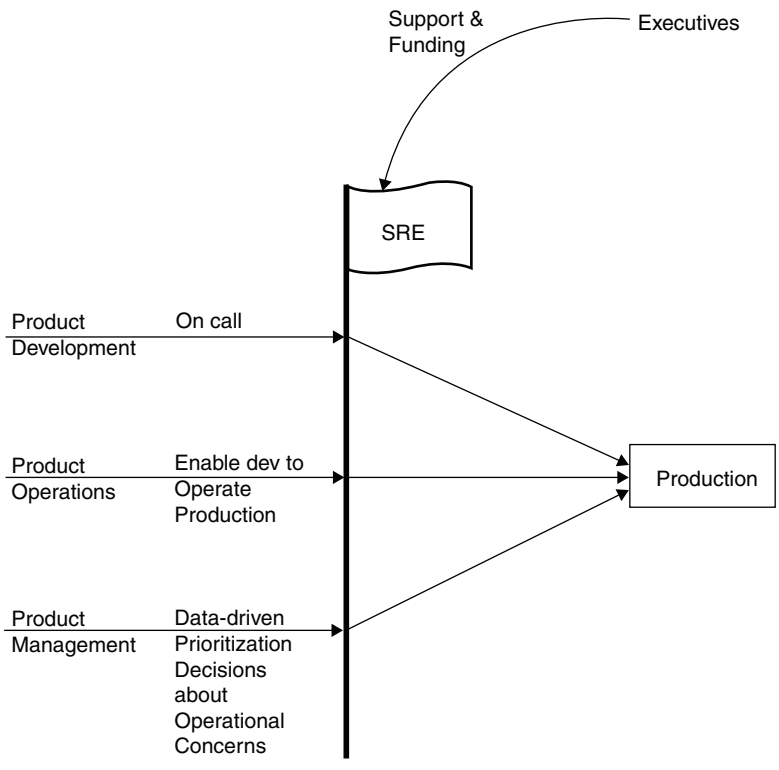


Figure 2.5 *Collective ownership of production operations using SRE*

SRE aligns the product delivery organization on operational concerns under its flag. Product development contributes to production operations by going on call to get firsthand experience with how the product meets customer demands in production. This experience is fed into the new feature and infrastructure development. The result is a maximization of feature development time while ensuring that the product meets customer demands in production.

Product operations makes a contribution by enabling developers to do production operations themselves. This is done through development of the SRE infrastructure as a framework to be used by the developers.

Product management contributes by making data-driven prioritization decisions about the most important user journeys for which the incidents need to be detected. Further, the contribution is in agreements with the autonomous incident backlog prioritization by the people on call. Additionally, the contribution is to be aligned with the data-driven prioritization decisions about reliability enablers to be included in the user story backlog.

The SRE transformation does not come for free. An investment in time, money, and effort is required to align the product delivery organization on operational concerns using SRE. That is why the executives need to also get involved. The executives can contribute twofold to the SRE transformation. First, they need to support the topic. This can be done in all-hands meetings, smaller conversations, and one-on-one discussions. Executive communication regarding

SRE needs to clarify to everyone in the organization that the topic has executive support. This goes a long way toward creating alignment behind SRE at every level of the product delivery organization.

Moreover, the SRE transformation requires some slight investments in tooling and infrastructure. Although they are small scale, these investments need to be done in a timely manner so as not to impede the speed of transformation. In a large enterprise, it could take a significant amount of time to place orders due to supplier selection and data protection processes. Still, it is worth the effort. Voting with one's wallet is a good way for executives to underpin the verbal message of endorsing the SRE transformation.

2.4 The Challenge Statement

With an in-depth understanding of what to strive for during SRE transformation, the challenge statement can be concisely presented. Following is what the challenge is about.

SRE is an operations methodology that aligns the product delivery organization on operational concerns. The key challenge in traditional software delivery organizations is the misalignment on production operations. In such organizations, the following is true.

- Developers do not know why they should be doing operations.
- Operations engineers do not know why developers are not interested in operations.
- Product managers think operations work is done by operations engineers.
- Management does not promote and fund the topic.

In such a software delivery organization, there are no solid foundations on top of which SRE can be built as a practice. The foundations need to be put in place first. This will be a major part of the SRE transformation. The transformation will need to shift stakeholders' mindsets in the following ways.

- Developers should want to be involved in on-call processes to gain enough current operational knowledge to develop features that work well in production.
- Operations engineers should want to enable developers to perform service operations by providing the SRE infrastructure as a framework in order to distribute the operational work throughout the product delivery organization in an optimal way.
- Product managers should want to be involved in operations to help reduce customer escalations by making decisions based on production data. The decisions are about the prioritization of user journeys for which the incidents need to be detected, agreements on incident backlog handling, and prioritization of reliability features.
- Executives should want to enable effective and efficient product operations by promoting SRE and providing appropriate funding in a timely fashion.

Each party in the software delivery organization benefits from SRE. The benefits make it worth undergoing the SRE transformation. The benefits can be used as a beacon to aspire for, unleashing fun on the SRE transformation journey. This book will take you on the journey of transforming a software delivery organization bit by bit into one that does operations the SRE way and enjoys doing so. To get started, in the next section let us look at the general way the SRE transformation can be executed.

2.5 Coaching

A product delivery organization consists of many people who are organized in teams. The SRE transformation process has a clear goal to establish SRE as the central methodology for production operations in the teams. However, running the SRE transformation is not like running a project with predefined milestones to be tracked. Rather, the SRE transformation process is a network of induced changes and feedback on the changes cast in parallel at different teams and individuals. Many teams will be transforming at the same time, but the changes and feedback loops will be unique per team and individual. This is illustrated in Figure 2.6.

Now, how do you set up the SRE transformation process in the way shown in Figure 2.6?

One way to run the SRE transformation is by means of coaching. According to Wikipedia, "coaching is a form of development in which an experienced person, called a coach, supports a learner or client in achieving a specific personal or professional goal by providing training and

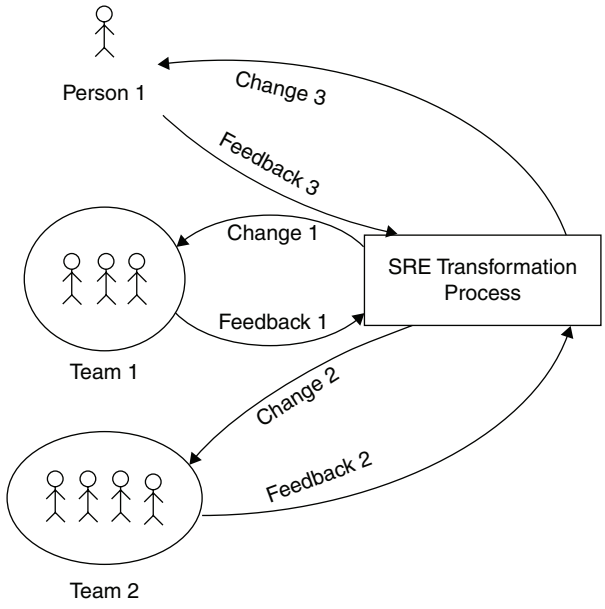


Figure 2.6 *SRE transformation process*

guidance."[3] It follows that coaching works with people and teams on an individual basis. This is the kind of approach needed for being empathetic and structured at the same time when running the transformation.

What makes coaching particularly interesting in the context of the SRE transformation is that it already exists as a discipline at both the organizational and team levels. According to the Institute of Coaching, "Organizational coaching aims at fostering positive, systemic transformation within organizations."[4] Within the broad theme of coaching, organizational coaching is a well-established discipline.

Team coaching, on the other hand, is a more recent and less structured discipline. It gained significance in the past decade. According to TPC Leadership, "team coaching is the art of facilitating and challenging a real team to maximize its performance and enjoyment in service of meaningful organizational goals."[5]

That is, coaching as a discipline with distinct subdomains of organizational coaching and team coaching is going to be a suitable approach for running the SRE transformation. Both types of coaching would be applied simultaneously and would work in tandem. However, when choosing organizational and team coaching as a methodology to run the SRE transformation, the question is, who would do the coaching? Who can act as a coach if the organization is new to SRE? Would external coaches be required? Would it be possible to develop internal coaches? If so, who would develop the internal coaches? Different approaches are possible here.

External coaches can be valuable to bring a fresh experience-based SRE perspective to an organization and quickly establish an understanding of SRE basics across the board. However, SRE coaches with experience doing successful SRE transformations are really difficult to find in the industry. This is because the first original SRE book by Google, *Site Reliability Engineering: How Google Runs Production Systems,* was only published in 2016.[6] Given that SRE transformations are taking several years to achieve in larger organizations, the pool of available coaches is going to be rather small.

Another aspect is that precisely because an SRE transformation takes several years to run in a larger organization, employing external coaches for the transformation time frame is hardly a financially viable option. It follows that coaches for SRE transformation need to be found and grown internally.

Which options would the coaches have to learn about SRE and bring themselves to a level that is necessary to coach others? Both the aforementioned Google book *Site Reliability Engineering* and Google's *The Site Reliability Workbook: Practical Ways to Implement SRE*[7] are a great and necessary starting point. These books will show potential coaches what needs to be done to achieve SRE with the sophistication and scale of Google. Additional books by former Googlers—*Implementing Service Level Objectives: A Practical Guide to*

3. Wikipedia. 2021. "Coaching." https://en.wikipedia.org/wiki/Coaching.

4. David, Susan. 2015. "Introduction to Organizational Coaching." Institute of Coaching, January 13, 2015. https://instituteofcoaching.org/resources/introduction-organizational-coaching.

5. Cardillo, Andrea. 2019. "How Is Team Coaching Different from Group Coaching?" TPC Leadership, July 10, 2019. https://tpcleadership.com/how-is-team-coaching-different-from-group-coaching.

6. Murphy, Niall Richard, Betsy Beyer, Chris Jones, and Jennifer Petoff. 2016. *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA: O'Reilly Media.

7. Beyer, Betsy, Niall Richard Murphy, David K. Rensin, Stephen Thorne, and Kent Kawahara. 2018. *The Site Reliability Workbook: Practical Ways to Implement SRE*. Sebastopol, CA: O'Reilly Media.

SLIs, SLOs, and Error Budgets[8] and *Real-World SRE: The Survival Guide for Responding to a System Outage and Maximizing Uptime*[9]—provide additional in-depth experience perspectives on SRE.

It is our aspiration that this book in particular will show potential coaches how SRE can be put in place in an organization that has never done operations the SRE way before. Further, the coaches can network with others at relevant conferences and industry events. Two conferences can be of special interest: SRECon[10] by USENIX and the DevOps Enterprise Summit[11] by IT Revolution. These conferences can be a great place to develop relationships with others practicing SRE or running SRE transformations. These relationships might lead to opportunities to visit other companies that are further along in the SRE journey. Seeing is believing, and seeing another company running the SRE process in a sophisticated manner can significantly boost one's own transformation.

Finally, the coaches can learn while running the SRE transformation in their own organization. In a larger organization, teams will inevitably adopt SRE at different speeds. Taking the learnings from the teams that lead the SRE transformation and transporting them to the teams catching up is a very valuable part of coaching. It enables the coach to gain experience and the teams to learn from each other.

From the Trenches: Long-lasting team-based coaching[12] that includes all team members—product owners, architects, developers, operations engineers and, at times, designers—is the most effective way to run the SRE transformation at the team level.

2.6 Summary

Introducing SRE requires changes in product operations, product development, and product management. The biggest change in product operations is the development of the SRE infrastructure as a framework that enables developers to go on call and operate their services in production. The biggest change in product development is to actually get involved in on call and operate the services developed in a real production environment. The delineation of how much developers are on call compared to how much operations are on call will vary by organization.

The challenge of the SRE transformation is that in a traditional software delivery organization, the product operations team has never provided frameworks enabling others to do operations work. Likewise, product development has never done operations work. Thus, there is

8. Hidalgo, Alex. 2020. *Implementing Service Level Objectives: A Practical Guide to SLIs, SLOs & Error Budgets*. Sebastopol, CA: O'Reilly Media.

9. Welch, Nat. 2018. *Real-World SRE: The Survival Guide for Responding to a System Outage and Maximizing Uptime*. Birmingham, UK: Packt Publishing Ltd.

10. "SRECon." 2017. USENIX. August 25, 2017. https://www.usenix.org/srecon.

11. IT Revolution Events. n.d. "DevOps Enterprise Summit 2022." Accessed January 12, 2022. https://events.itrevolution.com.

12. Guendisch, Philipp, and Vladyslav Ukis. 2022. "Employing Agile Coaching to Establish SRE in an Organization." *InfoQ*, August 23, 2022. https://www.infoq.com/articles/establish-SRE-coaching.

a lack of foundation on which SRE can be established. Developers do not understand why they should be doing operations. Operations engineers do not provide frameworks to enable developers to do operations. Managers do not promote the topic, let alone fund it. To drive the SRE transformation throughout the organization, SRE coaches need to be developed and designated.

In the next chapters, the journey of transforming a software delivery organization toward SRE will unfold. To get started, we will learn the basic SRE concepts in the next chapter.

Index

Numerics

100% availability, 50–52

A

Adzic, G., *Fifty Quick Ideas to Improve Your User Stories*, 99
agile/agility
  coach, 272
  delivery, 1
  enterprise-level, 318
AIDA marketing funnel, 92
  agreement, 95–96
  awareness, 92
  desire/understanding, 93–94
  engagement, 96
  interest, 93–94
alert(s), 25, 34, 108, 152, 166–167, 168–169. *See also* reacting to alerts on SLO breaches
  deduplication, 246–248
  dispatching, 211–212
    escalation policy, 212–215
    stakeholder groups, 216–217
    stakeholder notifications, 216, 217–219, 222–226
    stakeholder rings, 219–222
  error budget, 54
  escalation, 212–214
  mapping to incident priorities, 240–241
alignment
  mis-, 22–30
  organizational, 67–68
  SRE, 23–17, 29, 100
    on-call rotation and, 62
    concept pyramid and, 59–63
    error budget policies and, 62–63
    SLOs, 60–62
allocating SRE capacity to a team decision workflow, 380–381
Allspaw, J., 253
  The Future of Monitoring, 421
API, 33, 363–366
  application performance management facility, 149–150. *See also* infrastructure
automation
  DevOps and, 6
  incident response, 260–262
availability, 148, 149, 150–151, 159–160. *See also* SLI(s) (service level indicator/s)
  error budget, 49, 50–52
  newsletter, 476
  SLO, 17
  targets, 194
awareness, ways of promoting, 92
  availability newsletter, 476
  broadcasting, 478–479
  CoP (community of practice), 475
  long-form wiki articles, 477–478
  SRE column in the engineering blog, 477
  SRE minutes, 475–476
  success stories, 172–173, 208–209, 226–227

B

backlog, 30
  incident detection and resolution, 36
  product development, 25–26, 29
BDD (behavior-driven development), 336
Blank-Adelman, D. N., *Seeking SRE: Conversations About Running Production Systems at Scale*, 393
bottleneck analysis, 482–483
bottom-up buy-in, 90, 117
  development team, 119–121
  operations team, 117–119
bridging, 78
broadcasting
  service status, 299
  success stories, 302
  system, 478–479

C

"The Calculus of Service Availability", 372
CALMS (Culture, Automation, Lean, Measurement, and Sharing), 6
champions, 167
chaos engineering hypotheses selection workflow, 383–388
"Chaos Engineering: the history, principles, and practice", 388
chart, stakeholder, 100–103. *See also* graphs; tables; visualizations
CIOs (chief information officers), operations methodologies, 15–19
coach(ing), 39–40, 65–67, 96, 107, 115, 122, 123, 127–129. *See also* team(s)
  agile, 272, 315–316
  assessing transformation readiness, 137
  buy-in and, 90–91
  champions and, 167
  conveying the basics, 136–137
    on call for SLO breaches, 146–147
    causes of SLO breaches, 140–145
    motivation to fix SLO breaches, 142–144
    SLO as a contract, 137–138
    SLO as a proxy measure of customer happiness, 138
    SLO versus technical monitoring, 144–145
    user personas, 138–140
    user story mapping, 140–142
  dealing with detractors, 168–171
  implementing transformation in all teams, 129–131
  organizational, 131–133
  qualities, 97–98
  responsibilities, 98–99
  SRE transformation meeting and, 112–114
  support system, 487–489
  teaching the log query language, 156–157
  team, 124–125, 126–127
  COBIT (Control Objectives for Information and Related Technologies), 4–5, 8–23, 27–17
Code Red, 322
Code Yellow, 322
collective ownership, 67
  of production operations, 23–32
    benefits and costs, 36–38
    product development and, 25–27
    product management and, 32–36
    product operations, 28–35
complex incident(s), 248. *See also* incident(s)
  classification, 250, 252–253
  existing incident coordination systems, 249–250
  LFI (Learning from Incidents in Software) community, 253
  response
    on-call roles, 257–258
    process diagram, 260–262
    process evaluation, 258–259
    responsibilities, 256
    roles, 254–256
    team well-being, 262–268
  severity, 250, 251–252, 253–254, 257
concept pyramid, 55–59, 130, 136, 147, 305, 329, 491
  error budget, 57–58
  reliability stack, 55–56
  SLIs, 56
  SLOs, 56
  SRE alignment, 59–63
conferences, 40
  DevOps Enterprise Summit, 487
  QCon, 487
  SREcon, 487
Continuous Delivery, 6, 317
  indicators
    bottleneck analysis, 482–483
    combining with SRE indicators, 479–480
    versus SRE indicators, 481
Cook R., LFI (Learning from Incidents in Software), 253
cooperation, status quo and, 75–77
CoP (community of practice), 206–208, 475
correctness, 44
cost accounting, SRE team, 417–419
cost optimization
  domain rotation, 425
  thin-layer on-call rotation, 426
Covey, S. R., *The Speed of Trust: The One Thing that Changes Everything*, 458
CPOs (chief product officers), operations methodologies, 15–19
CRE (Customer Reliability Engineering), 102
cross-pair rotation, 183
CTOs (chief technical officers), operations methodologies, 15–19
culture
  generative, 75–76
  organizational, 74–75, 89
customer(s). *See also* user
  escalation, 35–36
  incident response and, 252–253
  support ticket trend indicator, 349–350
    graph representation, 350–351
    table representation, 352–353

D

dashboards, SLI/SLO, 165–166
decision-making
  error budget-based, 57–58, 325
    "On-Call Rotations by Team" indicator, 353–354
    customer support ticket trend indicator, 349–353
    dimensions of SRE indicators, 330
    implementing SRE indicators, 330
    incident time to recovery trend indicator, 354–356
    least available service endpoints indicator, 356–357
    premature SLO error budget exhaustion indicator, 339–343
    reliability decision-making taxonomy, 325–329
    SLA adherence indicator, 348–349
    SLA error budget depletion indicator, 345–348
    "SLAs by Service" indicator, 343–344
    SLO adherence indicator, 332–333
    SLO error budget depletion indicator, 333–339
    "SLOs by Service" indicator, 330–332
    slowest service endpoints indicator, 358
  incident response, 255
  versus indicators, 359–362
  joint, 105–106
  workflows, 362–363
    allocating SRE capacity to a team decision, 380–381
    API consumption, 363–366
    chaos engineering hypotheses selection, 383–388
    features versus reliability prioritization, 368–372
    setting an SLA, 377–380
    setting an SLO, 372–377
    tightening a dependency's SLO, 366–368
default SLOs, 163–164
defining
  escalation policy, 214–215
  generic incident priorities, 234–237
  hypotheses, 484–486
  incident priorities, 244
  SLOs, 158
  stakeholder notifications, 224
defining an SLO, 161
detractors, 168–171
DevOps, 1, 2, 6, 25, 33, 180
  CALMS, 6
  comparison with other operations methodologies, 8–23
  Enterprise Summit, 40, 487
  head of development, 106–107
  implementation, 35
  pillars of success, 6
  in the "who builds it, who runs it?" spectrum, 441–442
Dickerson, M., *Site Reliability Engineering: How Google Runs Production Systems*, 71
documentation
  application performance management facility, 149–150
  creating, 171–172
  incident response, 301–302
  log query language, 156
  runbooks, 203–205
domain rotation, 425. *See also* on-call work
durability, 44

E

engagement, 96. *See also* buy-in
enterprise
  -level agility, 318
  radical, 431–432
error budget(s), 47–49, 57
  alerts, 54
  availability SLO and, 49
  -based decision making, 325
    allocating SRE capacity to a team decision workflow, 380–381
    API consumption decision workflow, 363–366
    "On-Call Rotations by Team" indicator, 353–354
    chaos engineering hypotheses selection workflow, 383–388
    customer support ticket trend indicator, 349–353
    dimensions of SRE indicators, 330
    features versus reliability prioritization workflow, 368–372
    implementing SRE indicators, 330
    incident time to recovery trend indicator, 354–356
    least available service endpoints indicator, 356–357
    premature SLO error budget exhaustion indicator, 339–343
    reliability decision-making taxonomy, 325–329
    setting an SLA decision workflow, 377–380

setting an SLO decision workflow, 372–377
SLA adherence indicator, 348–349
SLA error budget depletion indicator, 345–348
"SLAs by Service" indicator, 343–344
SLO adherence indicator, 332–333
SLO error budget depletion indicator, 333–339
"SLOs by Service" indicator, 330–332
slowest service endpoints indicator, 358
tightening a dependency's SLO decision workflow, 366–368
Code Red, 322
Code Yellow, 322
debt, 55
decision-making, 57–58
latency, 52
logging, 154–155
mapping to incident priorities, 239–240
policy, 53–55, 57, 305
agreeing to, 318–319
clause, 312–313
conditions, 309–310
consequences, 311
effectiveness, 314
enacting, 320
extending, 314–318
feature development and, 305–306
governance, 312–314
motivation for, 305–306
reliability and, 306
review, 309
reviewing, 321
scope, 308
storing, 319–320
structure, 308–309
template, 308
value, 313
premature depletion, 307
silver bullet, 322
thaw tax, 323
top-up, 323
of zero, 50–52
escalation policy, 212–215, 258, 308–309
Evans, D., *Fifty Quick Ideas to Improve Your User Stories*, 99
experiments, chaos, 383–385

F

failures, 78
feature development, 26, 32
backlogs, 30
error budget policy, 305–306
versus reliability prioritization workflow, 368–372
feedback
customer, 394
loops, 483–484
postmortem, 282, 288–289
fidelity, 44
focus-based working mode, 185
formal leadership, 69
Forsgren, N., *Accelerate*, 103–104
forums, 94, 206, 283
frameworks, 34. *See also* COBIT (Control Objectives for Information and Related Technologies); DevOps; ITIL; operations methodologies
COBIT, 4–5
ITIL, 3–4
Scaled Agile, 315–316
freshness, 44
funding, SRE team, 414–417

G

game days, 476–478
Gates, B., *The Road Ahead*, 491
generic incident priorities, 234–237
GitHub, 289–290, 293
Goodhart's law, 359
Google, 2
CRE (Customer Reliability Engineering), 102
online searches, 25
SRE (Site Reliability Engineering). *See* SRE (Site Reliability Engineering)
governance, error budget policy, 312–314
graphs, 152–154
customer support ticket trends, 350–351
error budget depletion, 333–335, 345–347
premature error budget exhaustion, 339–341

H

Hand, J., 43
Helfand, H., *Dynamic Reteaming: The Art and Wisdom of Changing Teams*, 267
Hohpe, G., *The Software Architect Elevator: Redefining the Architect's Role in the Digital Universe*, 437
holacracy, 431
HTTP response codes, availability SLO and, 150–151
Humble, J., 6
Accelerate, 103–104
hypotheses, 18, 107, 159

chaos engineering and, 383–388
defining, 484–486
posing, 81–84
testing, 467–469

I

ICS (Incident Command System), 249
identity
team, 413–414
triangle, 429–431
Implementing Service Level Objectives: A Practical Guide to SLIs, SLOs, and Error Budgets, 40
incident(s), 35, 51, 229–230
backlog, 36
classification, 250, 252–253
complex, 248
deduplication, 246–248
game days, 476–478
infrastructure, 245
LFI (Learning from Incidents in Software) community, 253
manual creation, 260
MTTR
MTBF and, 420–421
versus reliability user experience, 420
postmortem, 77, 268–269. *See also* postmortem
action items, 283, 284–285
analyzing recorded conversations, 274
content consumption statistics, 285–288
criteria, 269–271
distribution of findings, 280–282
drafting the write-up, 274
examples, 292–293
facilitating learning from, 291
generating action items, 278–280
initiating, 271–272
internal write-up, 283
lifecycle, 272–273
participant feedback, 282
practice measurements, 291–292
prime directive, 276–277
reviewing the timeline, 277
soliciting feedback on outcomes, 288–289
taking care of people issues, 274–276
template, 289–290
timeline reconstruction, 273
priorities, 230, 253–254
adjusting based on stakeholder feedback, 242–244
changing during an incident, 233–234
defining, 244
generic, 234–237
mapping to alerts, 240–241
mapping to error budgets, 239–240
mapping to SLOs, 237–239
uncovering new use cases, 242
recovery time, 354–356
reducing customer escalations, 35–36
response
broadcasting successes, 302
documenting, 301–302
on-call roles, 257–258
process diagram, 260–262
process evaluation, 258–259
responsibilities, 256
roles, 254–256
team well-being, 262–268
transparency and, 298
service status page, 298–301
severity, 250, 251–252, 253–254, 257
SLO breaches and, 232–233
timeline, 36
informal leadership, 68
information sharing, 131–133
infrastructure, 34, 127–129, 147–148, 164, 495–496. *See also* alert(s); SLI(s) (service level indicator/s); SLO(s) (service level objective/s)
alerts, 166–167
application performance management facility, 149–150
dashboards, 165–166
incident response, 245
logging, 154–155
prioritization of features, 152–154, 167
SRE indicators, 330
innovation, status quo, 78–79
insider knowledge. *See also* knowledge sharing
on-call rotation, 32–33
product operations, 29, 33
software development, 33–34
interest, 93–94
internal Stack Overflow, 205–206
inter-pair rotation, 183
interruption-based working mode, 181–184
IT organizations, 437
iterating on an SLO, 159–162
ITIL, 3–4, 27–17
comparison with other operations methodologies, 8–23
incident response process, 229–230

J

Jeli, 274
joint
  buy-in, 112–114
  decision-making, 105–106
Jones, N., Learning from Incidents in Software (LFI) community, 253

K

Kanban board, visualizing SRE transformation, 132–133
Kim, G., *Accelerate*, 103–104
knowledge sharing, 196–197
  needs, 198–199
  pyramid, 199–201
  runbooks, 203–205
"Known Known", 387
"Known Unknown", 387
KPIs (key performance indicators)
  SRE indicators and, 359
  SRE team, 419–421

L

latency, 43, 52, 148, 149, 151–152, 160–161
lateral buy-in, 90, 122–123
leadership, 68. *See also* buy-in
  formal, 69
  head of development, engaging
    assessing cost and effort, 106–107
    buy-in, 103
    DevOps best practices, 106–107
    improving relationships, 104–106
  head of operations, engaging, 107–108
    assessing cost and effort, 109–110
    improving on-call setup, 108
    improving the state of production reporting, 108
  head of product management, engaging, 110–112
  informal, 68
  joint decision-making, 105–106
  stakeholder chart, 100–103
Lean, 6
learning opportunities, 486–487
least available service endpoints indicator, 356–357
LFI (Learning from Incidents in Software) community, 253
log query languages, 156–157
logging, 34–29, 74, 154–155
long-form wiki articles, 477–478

M

marketing, AIDA, 92
  agreement, 95–96
  awareness, 93
  desire/understanding, 93–94
  engagement, 96
  interest, 93–94
maturity model, 81
McGhee, S., 49, 373
measurement, 6. *See also* metrics; SLI(s) (service level indicator/s); SLO(s) (service level objective/s)
  availability, 150–151
  error budget policy effectiveness, 314
  latency, 151–152
  logging and, 154–155
  postmortem practice, 291–292
meetings
  incident response, documenting, 301–302
  postmortem
    distribution of findings, 280–282
    generating action items, 278–280
    participant feedback, 282
    prime directive, 276–277
    reviewing the timeline, 277
  SRE transformation, 112–114
metrics, 102, 103–104, 105
  postmortem content consumption, 285–288
  transformation
    executives' perception, 471–472
    reliability perception by users and partners, 472–473
    services exhausting error budgets prematurely, 470–471
    undetected outages, 469–470
middle management, lateral buy-in, 122–123
misalignment, product delivery and, 22–30
mobile devices, 297
modeling, 5–6, 8–23
monitoring, SLOs and, 144–145
movement, 89–90. *See also* awareness, ways of promoting
MTBF, 420–421
MTTR
  MTBF and, 420–421
  versus reliability user experience, 420

N

"Nonviolent Communication" approach, 275–276
"not-invented-here" syndrome, 11
novelty, 78–79. *See also* innovation
NPS (Net Promoter Score), 472–473
Nygard, M. T., *Release It!: Design and Deploy Production-Ready Software*, 338

O

on-call work, 182, 183
  incident response, 257–258
  knowledge sharing, 196–197
    needs, 198–199
    pyramid, 199–201
  management tools, 188, 190–193, 246, 294
    connecting to, 294–296
    status page capability, 298–301
    subscribing to stakeholder notifications, 225–226
  out-of-hours, 193
  rotation, 32–33, 185
    cross-pair, 183
    inter-pair, 183
    one person on call, 186–187
    scheduling, 186
    thin-layer, 426
    three people on call, 187–188
    two people on call, 187
  runbooks, 203–205
  scheduling, 190–191
  setup, 108
  SLO breaches and, 143–144, 146–147
  SRE alignment and, 62
  trade-offs, 194–196
  training, 198, 200, 201–203
  using availability targets and product demand, 194
online searches, Google, 25
operations methodologies. *See also* COBIT (Control Objectives for Information and Related Technologies); DevOps; ITIL; modeling; SRE (Site Reliability Engineering)
  comparing, 8–23
organizational. *See also* product delivery organization
  alignment, 67–68
  coaching, 131–133
  culture, 74–75, 89, 256
  structure, 65–67, 391. *See also* "who builds it, who runs it?" spectrum
    questions driving, 391–392
    versus SRE principles, 393–394
organizational buy-in, 89–91
  bottom-up, 90, 117
    development team, 119–121
    operations team, 117–119
  joint, 112–114
  lateral, 90, 122–123
  staggering, 123
  top-down, 90, 99. *See also* top-down buy-in
    assessing cost and effort, 106–107
    head of development, engaging, 103–107
    head of operations, engaging, 107–110
    head of product management, engaging, 110–112
    metrics and, 103–104, 105
    stakeholder chart, 100–103
outages, strategies for coping with, 244. *See also* incident(s)
out-of-hours on-call, 193
ownership. *See also* collective ownership
  collective, 23–32, 67
  of production operations
    in product development, 25–27
    in product management, 32–36
    in product operations, 28–35

P

PagerDuty, 249, 250, 268
Parker, M., *A Radical Enterprise: Pioneering the Future of High-performing Organizations*, 431–432
partial ownership, of production operations
  in product development, 25–27
  in product management, 32–36
  in product operations, 28–35
people. *See also* buy-in
  communication and, 274–276
  movements, 89–90
  in the product delivery organization, 69–71
PI (Program Increment), 315
policy
  error budget, 53–55
    agreeing to, 318–319
    clause, 312–313
    conditions, 309–310
    consequences, 311
    effectiveness, 314
    enacting, 320

extending, 314–318
feature development, 305–306
governance, 312–314
motivation, 305–306
reliability, 306
review, 309
reviewing, 321
scope, 308
storing, 319–320
structure, 308–309
template, 308
value, 313
escalation, 212–215, 258, 308–309
portfolio management, SRE transformation and, 114–116, 131
posing hypotheses, 81–84
posting SLO breaches, 188–190
postmortem, 77, 244, 268–269
action items, 283
generating, 278–280
lead and cycle times, 284–285
analyzing recorded conversations, 274
content consumption statistics, 285–288
criteria, 269–271
distribution of findings, 280–282
drafting the write-up, 274
examples, 292–293
facilitating learning from, 291
initiating, 271–272
internal write-up, 283
participant feedback, 282
practice measurements, 291–292
prime directive, 276–277
reviewing the timeline, 277
soliciting feedback on outcomes, 288–289
taking care of people issues, 274–276
template, 289–290
timeline reconstruction, 273
premature
error budget depletion, 307
SLO error budget exhaustion indicator, 339
graph representation, 339–341
table representation, 341–343
presentations, 95–96
introduction to SRE transformation, 135–136
slide decks, awareness-related, 93
principles
COBIT, 5
ITIL 4
SRE, 7–11
processes, 79–80, 122
incident response
dynamics, 260–262
evaluating, 258–259
SRE transformation and, 316–317
product delivery organization, 2, 11, 15. *See also* DevOps; SRE (Site Reliability Engineering)
alignment, 29, 67–68
assessing the status quo
bridging, 78
cooperation, 75–77
culture, 74–75
formal leadership, 69
handling failures, 78
informal leadership, 68
innovation, 78–79
leadership, 68
people, 69–71
process, 79–80
risk sharing, 77
structure, 65–67
technology, 71–74
misalignment, 22–30
operations methodologies, 16–23
product operations, 29
SRE alignment, 23–17
SRE buy-in, 90–91
product development, 29, 106–107. *See also* DevOps
backlogs, 25–26
buy-in, 119–121
on-call rotation, 32–33
head of, engaging
buy-in and, 103–104
improving relationships, 104–106
incident detection and resolution, 35–36
learning the log query language, 156–157
partial ownership of production operations, 25–27
team coaching, 126–127
product management, 30, 32
head of, engaging, 110–112
partial ownership of production operations, 32–36
product operations, 1, 29. *See also* DevOps; SRE (Site Reliability Engineering)
alerts, 25
buy-in, 117–119
collective ownership, 23–32
comparison of methodologies, 8–23
head of, engaging, 107–108
assessing cost and effort, 109–110
improving on-call setup, 108

improving the state of production reporting, 108
head of product management, engaging, 110–112
insider knowledge, 29, 33
partial ownership of production operations, 28–35
team coaching, 126
product owners, team coaching, 124–125
production, 30
on-call rotation, 32–33
collective ownership, 67
benefits and costs, 36–38
product development and, 25–27
product management and, 32–36
product operations, 28–35
deployment, 59
fixing issues in, 25–15, 29
processes, 79–80
service reliability hierarchy, 71–72
testing, 32
professional on-call management tools, 191–193

Q-R

QCon, 487
radical enterprise, 431–432
reacting to alerts on SLO breaches, 175, 177. *See also* on-call work
environment selection, 175–177
internal Stack Overflow, 205–206
on-call rotation, 185, 197
knowledge sharing needs, 198–199
one person on call, 186–187
scheduling, 186
systematic knowledge sharing, 196–197
three people on call, 187–188
two people on call, 187
operational responsibilities, 178–179
versus development responsibilities, 177–178
splitting, 179–180
out-of-hours on-call, 193
responsibilities, 177
runbooks, 203–205
working modes, 180–181
focus-based, 185
interruption-based, 181–184
Real-World SRE: The Survival Guide for Responding to a System Outage and Maximizing Uptime, 40
regulatory compliance, 27–17, 316, 494–495
release lifecycle phases, 315
reliability, 44, 115. *See also* error budget(s)
error budget policy, 306
error budget-based decision making, 325–329
versus features prioritization workflow, 368–372
service, 71–72
user experience, 111
resiliency, 51
resource-based alerts, mapping to incident priorities, 240–241. *See also* alert(s)
revising SLOs, 162
risk sharing, 77
rollouts, 316
runbooks, 225

S

SAFe (Scaled Agile Framework), 315–316
Safety II perspective, 253
scheduling, on-call work, 186, 190–191
scientific method, 18. *See also* hypotheses
Scrum of Scrums, 94
security, modeling, 6
self-service, 171–172
documentation, 171–172
SLO setting and editing, 161, 164
teaching and learning, 156–157
service
catalog, 492–493
reliability, 71–72
service status page, 298–301
Shook, J., "How to Change a Culture: Lessons from NUMMI", 79
silver bullet, 322
Site Reliability Engineering: How Google Runs Production Systems, 40, 43, 45, 71, 269
Site Reliability Workbook, 1–2, 7, 40, 112–115, 305–306, 393, 472
SLA(s) (service-level agreement/s), 101, 494
adherence indicator, 348–349
error budget depletion indicator, 345
graph representation, 345–347
table representation, 347–348
setting, 377–380
"SLAs by Service" indicator, 343–344
slide decks, awareness-related, 93
SLI(s) (service level indicator/s), 36, 43–44, 47–48
availability, 43
correctness, 44
dashboards, 165–166
durability, 44
fidelity, 44
freshness, 44

latency, 43, 151–152
logging, 154–155
reliability, 43
SRE concept pyramid and, 56
standardization, 147–149
system categories and, 45–44
user personas, 138–140
SLO(s) (service level objective/s), 17, 18, 45–48
adherence indicator, 332–333
alignment and, 60–62
availability, 17
100%, 50–52
error budget and, 49
breaches, 17, 142–144, 161–162. *See also* incident(s); reacting to alerts on SLO breaches
alerts, 152, 166–167
on call for, 146–147
causes of, 145–140
deduplication and, 246–248
incidents and, 232–233
log query language and, 156–157
posting, 188–190
visualizations, 152–154
as a contract, 137–138
dashboards, 165–166
default, 163–164
defining, 158, 161
error budget depletion indicator, 333
graph representation, 333–335
table representation, 335–339
"good", 157–158
iteration, 159–162
latency, 151–152
logging, 154–155
mapping to incident priorities, 237–239
as a proxy measure of customer happiness, 138
revising, 162
setting, 372–377
SRE concept pyramid and, 56
versus technical monitoring, 144–145
tightening a dependency's decision workflow, 366–368
user personas, 138–140
"SLOs by Service" indicator, 330–332
Sloss, B. T., 27, 34
slowest service endpoints indicator, 358
Smith, S., 395
"Aim for Operability, not SRE as a Cult", 416
"Implementing You Build It You Run It at Scale", 194
software development, 1. *See also* DevOps; product development
alerts, 34
backlogs, 25–26
on-call rotation, 32
feature development, 26
insider knowledge, 33–34
logging, 34–29
SRE (Site Reliability Engineering), 1–2, 7
AIDA marketing funnel, 92
agreement, 95–96
awareness, 92
desire/understanding, 93–94
engagement, 96
interest, 93–94
alignment, 23–17, 100
on-call rotation and, 62
concept pyramid and, 59–63
error budget policies and, 62–63
organizational, 67–68
SLO, 60–62
buy-in, 89–91. *See also* buy-in
bottom-up, 90, 117
joint, 112–114
lateral, 90, 122–123
staggering, 123
top-down, 90, 99
challenge statement, 38–39
coaches. *See also* coaches
qualities, 97–98
responsibilities, 98–99
COBIT, 4–5
collective ownership of production operations
benefits and costs, 36–38
product development and, 25–27
product management and, 32–36
product operations, 28–35
comparison with other operations methodologies, 8–23
concept pyramid, 55–59, 130, 136, 147, 305, 329, 491
error budget, 57–58
reliability stack, 55–56
SLIs, 56
SLOs, 56
CoP (community of practice), 206–208
DevOps, 6
error budget, 47–49. *See also* error budget(s)
alerts, 54
availability SLO and, 49
debt, 55

latency, 52
policies, 53–55
premature depletion, 307
of zero, 50–52
feedback loops, 483–484
generative culture, 75–76
identity triangle, 429–431
indicators, 330, 359
bottleneck analysis, 482–483
"On-Call Rotations by Team", 353–354
combining SRE with CD, 479–480
customer support ticket trend, 349–353
versus decisions, 359–362
dimensions, 330
Goodhart's law, 359
incident time to recovery trend, 354–356
KPIs, 359
least available service endpoints, 356–357
premature SLO error budget exhaustion, 339–343
SLA adherence, 348–349
SLA error budget depletion, 345–348
"SLAs by Service", 343–344
SLO adherence, 332–333
SLO error budget depletion, 333–339
"SLOs by Service", 330–332
slowest service endpoints, 358
SRE versus CD, 481
infrastructure, 34, 127–129, 147–148, 164, 495–496
alerts and, 166–167
dashboards, 165–166
incident response and, 245
logging, 154–155
prioritization of features, 152–154, 167
SRE indicators, 330
ITIL, 3–4
for large enterprise, 148
learning opportunities, 486–487
maturity model, 81
metrics, 103–104, 105
minutes, 475–476
modeling and, 5–6
movement, 89–90
organizations, 437–438
principles, 7–11, 393–394
reasons for using, 2
scientific method and, 18
SLIs (service level indicators), 36, 43–44
availability, 43, 150–151
correctness, 44
dashboards, 165–166
durability, 44
fidelity, 44
freshness, 44
latency, 43, 151–152
reliability, 43
system categories and, 45–44
user personas, 138–140
SLOs (service level objectives), 17, 18, 45–47
availability, 17
breaches, 17, 142–144, 145–140, 146–147, 161–162
as a contract, 137–138
dashboards, 165–166
default, 163–164
defining, 158
"good", 157–158
iteration, 159–162
mapping to incident priorities, 237–239
as a proxy measure of customer happiness, 138
revising, 162
user personas, 138–140
support for regulatory IT compliance, 27–17
transformation, 29, 31, 32, 35, 38, 65, 66, 68, 69, 71, 81, 89, 123. *See also* collective ownership; transformation
coaching and, 39–40
detractors, 168–171
hypotheses, 84
implementing in all teams, 129–131
introductory presentations, 135–136
leadership and, 98
meeting, 112–114
portfolio management and, 114–116
processes and, 316–317
stakeholders' exaggerated expectations, 83
team coaching and, 124–125
testing hypotheses, 467–469
visualizing, 132–133
ways of practicing, 440–441
wiki, 204, 238, 241, 245, 259
error budget policy and, 320
incident response documentation, 301–302
long-form articles, 477–478
SRE Weekly, 293
SREcon, 487
stability, 338
stakeholder
chart, 100–103
feedback, adjusting incident priorities based on, 242–244

groups, 216
notification(s), 216, 222–224, 225, 243
characteristics, 222–223
defining, 224
feedback questions, 224
subscribing, 225–226
targeting, 217–218
triggering, 218–219
rings, 219–222
subgroups, 216–217
standardization, SLI, 147–149
"State of DevOps 2021" report, 104
status quo, assessing, 66, 67–68
bridging, 78
cooperation, 75–77
handling failures, 78
innovation, 78–79
leadership, 68
formal, 69
informal, 68
organizational culture, 74–75
people, 69–71
process, 79–80
risk sharing, 77
structure, 65–67
technology, 71–74
storage, error budget policy, 319–320
Strathern, M., 359
subscribing, to stakeholder notifications, 225–226
success stories, broadcasting, 172–173, 208–209, 226–227, 302

T

tables
customer support ticket trends, 352–353
error budget depletion, 335–339, 347–348
premature error budget exhaustion, 341–343
team(s), 126. *See also* "who builds it, who runs it?" spectrum
bridging, 78
coaching, 39–40, 124–125, 126–129
cost accounting, 417–419
development
buy-in, 119–121
versus incident response, 268
introductory presentation, 136
service status broadcasting, 300
SLOs and, 137–138
funding, 414–417
head count allowance, 417
holacracy, 431
identity, 413–414
incentives, 412–413
incident response
versus development, 268
on-call roles, 257–258
responsibilities, 256
roles, 254–256
well-being, 262–268
KPIs, 419–421
operations
buy-in, 117–119
introductory presentation, 135
SRE transformation, 129–131
technical monitoring, versus SLO, 144–145
technology, 71–74
logging, 74
testing, 74
template
error budget policy, 308
postmortem, 289–290
runbook, 204
testing, 32, 74
hypotheses, 467–469
thaw tax, 323
thin-layer on-call rotation, 426
Thomas, Z., "The Smallest Possible SRE Team", 475
throughput, 44
tool(s)
connections, 296–297
internal Stack Overflow, 205–206
Jeli, 274
landscapes, 298
mashups, 296, 297
mobile devices, 297
on-call management, 190–193, 225–226, 294–296
status page capability, 298–301
top-down buy-in, 90, 99
head of development, engaging, 103
assessing cost and effort, 106–107
improving relationships, 104–106
metrics, 103–104, 105
head of operations, engaging, 107–108
assessing cost and effort, 109–110
improving on-call setup, 108
improving the state of production reporting, 108
head of product management, engaging, 110–112
stakeholder chart, 100–103
top-up, 323
training
on-call work, 198, 200, 201–203
SRE coaches, 489

transformation, 29, 31, 32, 35, 38, 65, 66, 68, 69, 71, 81, 89, 123. *See also* collective ownership
coaching and, 39–40
detractors, 168–171
hypotheses, 84, 467–469
implementing in all teams, 129–131
introductory presentations, 135–136
leadership and, 98
meeting, 112–114
metrics
executives' perception, 471–472
reliability perception by users and partners, 472–473
services exhausting error budgets prematurely, 470–471
undetected outages, 469–470
portfolio management and, 114–116
processes and, 316–317
service catalog and, 492–493
stakeholders' exaggerated expectations, 83
team coaching and, 124–125
visualizing, 132–133

U

UI/UX design, 110–111, 138, 140, 148
"Unknown Known", 388
"Unknown Unknown", 388
USENIX, 40
user
personas, 138–140
story mapping, 140–142

V

visualizations, 152–154, 241, 260. *See also* graphs; tables
visualizing, SRE transformation, 132–133
Vogels, W., "A Conversation with Werner Vogels," ACM Queue, 394

W

Westrum, R., "A Typology of Organisational Cultures", 75
"who builds it, who runs it?" spectrum, 395
choosing a model, 432
conveying the value to executives, 439–440
decision dimensions, 434–435
model transformation options, 433–434
positioning the SRE organization, 437–438
reporting options, 435–437
communicating the chosen model, 456–457
cost optimization, 424
domain rotation, 425
thin-layer on-call rotation, 426
DevOps role in, 441–442
introducing the chosen model, 457
model comparison, 400–403
reliability incentives, 397–399
reporting structure change options, 458–461
role changes, 461–462
SRE role, 440, 441, 442–443
assigning, 447–448
cultural importance of, 455
defining, 443–444, 445
fulfillment, 448–450
naming, 445–447
progressions, 451–453
responsibilities and skills, 445
transitions, 453–455
team topologies, 426–427
holacracy, 431
reporting lines, 427
SRE identity triangle, 429–431
ways of practicing SRE, 440–441
workers' council, 460
"you build it, ops run it" model, 398–399
"you build it, ops sometimes run it" model, 396
"you build it, SRE run it" model, 395, 421
SRE team in a dedicated SRE organization, 423–424
SRE team within a development organization, 422
SRE team within an operations organization, 423
"you build it, you and SRE run it" model, 395–396, 406
comparison of implementations, 411–412
price models for different SRE services, 418–419
SRE team cost accounting, 417–419
SRE team head count and budget, 414–417
SRE team in a dedicated SRE organization, 410–411
SRE team incentives, identity, and pride, 412–414
SRE team KPIs, 419–421
SRE team within the development organization, 406–408

SRE team within the operations organization, 409
"you build it, you run it but some specific use cases are additionally monitored by ops" model, 396
"you build it, you run it" model, 403–406
wiki, 204, 238, 241, 245, 259
error budget policy and, 320
incident response documentation, 301–302
long-form articles, 477–478
workflow, 362–363
allocating SRE capacity to a team decision, 380–381
API consumption decision, 363–366
chaos engineering hypotheses selection, 383–388
features versus reliability prioritization, 368–372
setting an SLA, 377–380
setting an SLO, 372–377
tightening a dependency's SLO decision, 366–368
working modes, 180–181
focus-based, 185
interruption-based, 181–184

X-Y-Z

"you build it" models. *See* "who builds it, who runs it?" spectrum