Preface

Software engineering has this in common with having children: the labor *before* the birth is painful and difficult, but the labor *after* the birth is where you actually spend most of your effort. Yet software engineering as a discipline spends much more time talking about the first period as opposed to the second, despite estimates that 40–90% of the total costs of a system are incurred after birth.[1] The popular industry model that conceives of deployed, operational software as being "stabilized" in production, and therefore needing much less attention from software engineers, is wrong. Through this lens, then, we see that if software engineering tends to focus on designing and building software systems, there must be another discipline that focuses on the *whole* lifecycle of software objects, from inception, through deployment and operation, refinement, and eventual peaceful decommissioning. This discipline uses—and needs to use—a wide range of skills, but has separate concerns from other kinds of engineers. Today, our answer is the discipline Google calls Site Reliability Engineering.

So what exactly is Site Reliability Engineering (SRE)? We admit that it's not a particularly clear name for what we do—pretty much every site reliability engineer at Google gets asked what exactly that is, and what they actually do, on a regular basis.

Unpacking the term a little, first and foremost, SREs are *engineers*. We apply the principles of computer science and engineering to the design and development of computing systems: generally, large distributed ones. Sometimes, our task is writing the software for those systems alongside our product development counterparts; sometimes, our task is building all the additional pieces those systems need, like backups or load balancing, ideally so they can be reused across systems; and sometimes, our task is figuring out how to apply existing solutions to new problems.

Next, we focus on system *reliability*. Ben Treynor Sloss, Google's VP for 24/7 Operations, originator of the term SRE, claims that reliability is the most fundamental feature of any product: a system isn't very useful if nobody can use it! Because reliability[2] is so critical, SREs are focused on finding ways to improve the design and operation of systems to make them more scalable, more reliable, and more efficient. However, we expend effort in this direction only up to a point: when systems are "reliable enough," we instead invest our efforts in adding features or building new products.[3]

Finally, SREs are focused on operating *services* built atop our distributed computing systems, whether those services are planet-scale storage, email for hundreds of millions of users, or where Google began, web search. The "site" in our name originally referred to SRE's role in keeping the *google.com* website running, though we now run many more services, many of which aren't themselves websites—from internal infrastructure such as Bigtable to products for external developers such as the Google Cloud Platform.

Although we have represented SRE as a broad discipline, it is no surprise that it arose in the fast-moving world of web services, and perhaps in origin owes something to the peculiarities of our infrastructure. It is equally no surprise that of all the post-deployment characteristics of software that we could choose to devote special attention to, reliability is the one we regard as primary.[4] The domain of web services, both because the process of improving and changing server-side software is comparatively contained, and because managing change itself is so tightly coupled with failures of all kinds, is a natural platform from which our approach might emerge.

Despite arising at Google, and in the web community more generally, we think that this discipline has lessons applicable to other communities and other organizations. This book is an attempt to explain how we do things: both so that other organizations might make use of what we've learned, and so that we can better define the role and what the term means. To that end, we have organized the book so that general principles and more specific practices are separated where possible, and where it's appropriate to discuss a particular topic with Google-specific information, we trust that the reader will indulge us in this and will not be afraid to draw useful conclusions about their own environment.

We have also provided some orienting material—a description of Google's production environment and a mapping between some of our internal software and publicly available

software—which should help to contextualize what we are saying and make it more directly usable.

Ultimately, of course, more reliability-oriented software and systems engineering is inherently good. However, we acknowledge that smaller organizations may be wondering how they can best use the experience represented here: much like security, the earlier you care about reliability, the better. This implies that even though a small organization has many pressing concerns and the software choices you make may differ from those Google made, it's still worth putting lightweight reliability support in place early on, because it's less costly to expand a structure later on than it is to introduce one that is not present. Management contains a number of best practices for training, communication, and meetings that we've found to work well for us, many of which should be immediately usable by your organization.

But for sizes between a startup and a multinational, there probably already is someone in your organization who is doing SRE work, without it necessarily being called that name, or recognized as such. Another way to get started on the path to improving reliability for your organization is to formally recognize that work, or to find these people and foster what they do—reward it. They are people who stand on the cusp between one way of looking at the world and another one: like Newton, who is sometimes called not the world's first physicist, but the world's last alchemist.

And taking the historical view, who, then, looking back, might be the first SRE?

We like to think that Margaret Hamilton, working on the Apollo program on loan from MIT, had all of the significant traits of the first SRE.[5] In her own words, "part of the culture was to learn from everyone and everything, including from that which one would least expect."

A case in point was when her young daughter Lauren came to work with her one day, while some of the team were running mission scenarios on the hybrid simulation computer. As young children do, Lauren went exploring, and she caused a "mission" to crash by selecting the DSKY keys in an unexpected way, alerting the team as to what would happen if the prelaunch program, P01, were inadvertently selected by a real astronaut during a real mission, during real midcourse. (Launching P01 inadvertently on a real mission would be a major problem, because it wipes out navigation data, and the computer was not equipped to pilot the craft with no navigation data.)

With an SRE's instincts, Margaret submitted a program change request to add special error checking code in the onboard flight software in case an astronaut should, by accident, happen to select P01 during flight. But this move was considered unnecessary by the "higher-ups" at NASA: of course, that could never happen! So instead of adding error checking code, Margaret updated the mission specifications documentation to say the equivalent of "Do not select P01 during flight." (Apparently the update was amusing to many on the project, who had been told many times that astronauts would not make any mistakes—after all, they were trained to be perfect.)

Well, Margaret's suggested safeguard was only considered unnecessary until the very next mission, on Apollo 8, just days after the specifications update. During midcourse on the fourth day of flight with the astronauts Jim Lovell, William Anders, and Frank Borman on board, Jim Lovell selected P01 by mistake—as it happens, on Christmas Day—creating much havoc for all involved. This was a critical problem, because in the absence of a workaround, no navigation data meant the astronauts were never coming home. Thankfully, the documentation update had explicitly called this possibility out, and was invaluable in figuring out how to upload usable data and recover the mission, with not much time to spare.

As Margaret says, "a thorough understanding of how to operate the systems was not enough to prevent human errors," and the change request to add error detection and recovery software to the prelaunch program P01 was approved shortly afterwards.

Although the Apollo 8 incident occurred decades ago, there is much in the preceding paragraphs directly relevant to engineers' lives today, and much that will continue to be directly relevant in the future. Accordingly, for the systems you look after, for the groups you work in, or for the organizations you're building, please bear the SRE Way in mind: thoroughness and dedication, belief in the value of preparation and documentation, and an awareness of what could go wrong, coupled with a strong desire to prevent it. Welcome to our emerging profession!

How to Read This Book

This book is a series of essays written by members and alumni of Google's Site Reliability Engineering organization. It's much more like conference proceedings than it is like a standard book by an author or a small number of authors. Each chapter is intended to be read as a part of a coherent whole, but a good deal can be gained by reading on whatever subject particularly interests you. (If there are other articles that support or inform the text, we reference them so you can follow up accordingly.)

You don't need to read in any particular order, though we'd suggest at least starting with Chapters The Production Environment at Google, from the Viewpoint of an SRE and Embracing Risk, which describe Google's production environment and outline how SRE approaches risk, respectively. (Risk is, in many ways, the key quality of our profession.) Reading cover-to-cover is, of course, also useful and possible; our chapters are grouped thematically, into Principles (Principles), Practices (Practices), and Management (Management). Each has a small introduction that highlights what the individual pieces are about, and references other articles published by Google SREs, covering specific topics in more detail. Additionally, the companion website to this book, *https://g.co/SREBook*, has a number of helpful resources.

We hope this will be at least as useful and interesting to you as putting it together was for us.

— The Editors

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`

Shows commands or other text that should be typed literally by the user.

`Constant width italic`

Shows text that should be replaced with user-supplied values or by values determined by context.

Tip

This element signifies a tip or suggestion.

Note

This element signifies a general note.

Warning

This element indicates a warning or caution.

Using Code Examples

Supplemental material is available at *https://g.co/SREBook*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Site Reliability Engineering*, edited by Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy (O'Reilly). Copyright 2016 Google, Inc., 978-1-491-92912-4."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Note

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of plans and pricing for enterprise, government, education, and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-998-9938 (in the United States or Canada)

707-829-0515 (international or local)

707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://bit.ly/site-reliability-engineering*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *https://www.oreilly.com*.

Find us on Facebook: *https://facebook.com/oreilly*

Follow us on Twitter: *https://twitter.com/oreillymedia*

Watch us on YouTube: *https://www.youtube.com/oreillymedia*

Acknowledgments

This book would not have been possible without the tireless efforts of our authors and technical writers. We'd also like thank the following internal reviewers for providing especially valuable feedback: Alex Matey, Dermot Duffy, JC van Winkel, John T. Reese, Michael O'Reilly, Steve Carstensen, and Todd Underwood. Ben Lutch and Ben Treynor Sloss were this book's sponsors within Google; their belief in this project and sharing what we've learned about running large-scale services was essential to making this book happen.

We'd like to send special thanks to Rik Farrow, the editor of *;login:*, for partnering with us on a number of contributions for pre-publication via USENIX.

While the authors are specifically acknowledged in each chapter, we'd like to take time to recognize those that contributed to each chapter by providing thoughtful input, discussion, and review.

Embracing Risk: Abe Rahey, Ben Treynor Sloss, Brian Stoler, Dave O'Connor, David Besbris, Jill Alvidrez, Mike Curtis, Nancy Chang, Tammy Capistrant, Tom Limoncelli

Eliminating Toil: Cody Smith, George Sadlier, Laurence Berland, Marc Alvidrez, Patrick Stahlberg, Peter Duff, Pim van Pelt, Ryan Anderson, Sabrina Farmer, Seth Hettich

Monitoring Distributed Systems: Mike Curtis, Jamie Wilkinson, Seth Hettich

Release Engineering: David Schnur, JT Goldstone, Marc Alvidrez, Marcus Lara-Reinhold, Noah Maxwell, Peter Dinges, Sumitran Raghunathan, Yutong Cho

Simplicity: Ryan Anderson

Practical Alerting from Time-Series Data: Jules Anderson, Max Luebbe, Mikel Mcdaniel, Raul Vera, Seth Hettich

Being On-Call: Andrew Stribblehill, Richard Woodbury

Effective Troubleshooting: Charles Stephen Gunn, John Hedditch, Peter Nuttall, Rob Ewaschuk, Sam Greenfield

Emergency Response: Jelena Oertel, Kripa Krishnan, Sergio Salvi, Tim Craig

Managing Incidents: Amy Zhou, Carla Geisser, Grainne Sheerin, Hildo Biersma, Jelena Oertel, Perry Lorier, Rune Kristian Viken

Postmortem Culture: Learning from Failure: Dan Wu, Heather Sherman, Jared Brick, Mike Louer, Štěpán Davidovič, Tim Craig

Tracking Outages: Andrew Stribblehill, Richard Woodbury

Testing for Reliability: Isaac Clerencia, Marc Alvidrez

Software Engineering in SRE: Ulric Longyear

Load Balancing at the Frontend: Debashish Chatterjee, Perry Lorier

Chapters Load Balancing in the Datacenter and Handling Overload: Adam Fletcher, Christoph Pfisterer, Lukáš Ježek, Manjot Pahwa, Micha Riser, Noah Fiedel, Pavel Herrmann, Paweł Zuzelski, Perry Lorier, Ralf Wildenhues, Tudor-Ioan Salomie, Witold Baryluk

Addressing Cascading Failures: Mike Curtis, Ryan Anderson

Managing Critical State: Distributed Consensus for Reliability: Ananth Shrinivas, Mike Burrows

Distributed Periodic Scheduling with Cron: Ben Fried, Derek Jackson, Gabe Krabbe, Laura Nolan, Seth Hettich

Data Processing Pipelines: Abdulrahman Salem, Alex Perry, Arnar Mar Hrafnkelsson, Dieter Pearcey, Dylan Curley, Eivind Eklund, Eric Veach, Graham Poulter, Ingvar Mattsson, John Looney, Ken Grant, Michelle Duffy, Mike Hochberg, Will Robinson

Data Integrity: What You Read Is What You Wrote: Corey Vickrey, Dan Ardelean, Disney Luangsisongkham, Gordon Prioreschi, Kristina Bennett, Liang Lin, Michael Kelly, Sergey Ivanyuk

Reliable Product Launches at Scale: Vivek Rau

Accelerating SREs to On-Call and Beyond: Melissa Binde, Perry Lorier, Preston Yoshioka

Dealing with Interrupts: Ben Lutch, Carla Geisser, Dzevad Trumic, John Turek, Matt Brown

Embedding an SRE to Recover from Operational Overload: Charles Stephen Gunn, Chris Heiser, Max Luebbe, Sam Greenfield

Communication and Collaboration in SRE: Alex Kehlenbeck, Jeromy Carriere, Joel Becker, Sowmya Vijayaraghavan, Trevor Mattson-Hamilton

The Evolving SRE Engagement Model: Seth Hettich

Lessons Learned from Other Industries: Adrian Hilton, Brad Kratochvil, Charles Ballowe, Dan Sheridan, Eddie Kennedy, Erik Gross, Gus Hartmann, Jackson Stone, Jeff Stevenson, John Li, Kevin Greer, Matt Toia, Michael Haynie, Mike Doherty, Peter Dahl, Ron Heiby

We are also grateful to the following contributors, who either provided significant material, did an excellent job of reviewing, agreed to be interviewed, supplied significant expertise or resources, or had some otherwise excellent effect on this work:

Abe Hassan, Adam Rogoyski, Alex Hidalgo, Amaya Booker, Andrew Fikes, Andrew Hurst, Ariel Goh, Ashleigh Rentz, Ayman Hourieh, Barclay Osborn, Ben Appleton, Ben Love, Ben Winslow, Bernhard Beck, Bill Duane, Bill Patry, Blair Zajac, Bob Gruber, Brian Gustafson, Bruce Murphy, Buck Clay, Cedric Cellier, Chiho Saito, Chris Carlon, Christopher Hahn, Chris Kennelly, Chris Taylor, Ciara Kamahele-Sanfratello, Colin Phipps, Colm Buckley, Craig Paterson, Daniel Eisenbud, Daniel V. Klein, Daniel Spoonhower, Dan Watson, Dave Phillips, David Hixson, Dina Betser, Doron Meyer, Dmitry Fedoruk, Eric Grosse, Eric Schrock, Filip Zyzniewski, Francis Tang, Gary Arneson, Georgina Wilcox, Gretta Bartels, Gustavo Franco, Harald Wagener, Healfdene Goguen, Hugo Santos, Hyrum Wright, Ian Gulliver, Jakub Turski, James Chivers, James O'Kane, James Youngman, Jan Monsch, Jason Parker-Burlingham, Jason Petsod, Jeffry McNeil, Jeff Dean, Jeff Peck, Jennifer Mace, Jerry Cen, Jess Frame, John Brady, John Gunderman, John Kochmar, John Tobin, Jordyn Buchanan, Joseph Bironas, Julio Merino, Julius Plenz, Kate Ward, Kathy Polizzi, Katrina Sostek, Kenn Hamm, Kirk Russell, Kripa Krishnan, Larry Greenfield, Lea Oliveira, Luca Cittadini, Lucas Pereira, Magnus Ringman, Mahesh Palekar, Marco Paganini, Mario Bonilla, Mathew Mills, Mathew Monroe, Matt D. Brown, Matt Proud, Max Saltonstall, Michal Jaszczyk, Mihai Bivol, Misha Brukman, Olivier

Oansaldi, Patrick Bernier, Pierre Palatin, Rob Shanley, Robert van Gent, Rory Ward, Rui Zhang-Shen, Salim Virji, Sanjay Ghemawat, Sarah Coty, Sean Dorward, Sean Quinlan, Sean Sechrist, Shari Trumbo-McHenry, Shawn Morrissey, Shun-Tak Leung, Stan Jedrus, Stefano Lattarini, Steven Schirripa, Tanya Reilly, Terry Bolt, Tim Chaplin, Toby Weingartner, Tom Black, Udi Meiri, Victor Terron, Vlad Grama, Wes Hertlein, and Zoltan Egyed.

We very much appreciate the thoughtful and in-depth feedback that we received from external reviewers: Andrew Fong, Björn Rabenstein, Charles Border, David Blank-Edelman, Frossie Economou, James Meickle, Josh Ryder, Mark Burgess, and Russ Allbery.

We would like to extend special thanks to Cian Synnott, original book team member and co-conspirator, who left Google before this project was completed but was deeply influential to it, and Margaret Hamilton, who so graciously allowed us to reference her story in our preface. Additionally, we would like to extend special thanks to Shylaja Nukala, who generously gave of the time of her technical writers and supported their necessary and valued efforts wholeheartedly.

The editors would also like to personally thank the following people:

Betsy Beyer: To Grandmother (my personal hero), for supplying endless amounts of phone pep talks and popcorn, and to Riba, for supplying me with the sweatpants necessary to fuel several late nights. These, of course, in addition to the cast of SRE all-stars who were indeed delightful collaborators.

Chris Jones: To Michelle, for saving me from a life of crime on the high seas and for her uncanny ability to find manzanas in unexpected places, and to those who've taught me about engineering over the years.

Jennifer Petoff: To my husband Scott for being incredibly supportive during the two year process of writing this book and for keeping the editors supplied with plenty of sugar on our "Dessert Island."

Niall Murphy: To Léan, Oisín, and Fiachra, who were considerably more patient than I had any right to expect with a substantially rantier father and husband than usual, for years. To Dermot, for the transfer offer.

[1]The very fact that there is such large variance in these estimates tells you something about software engineering as a discipline, but see, e.g., [Gla02] for more details.

[2]For our purposes, reliability is "The probability that [a system] will perform a required function without failure under stated conditions for a stated period of time," following the definition in [Oco12].

[3]The software systems we're concerned with are largely websites and similar services; we do not discuss the reliability concerns that face software intended for nuclear power plants, aircraft, medical equipment, or other safety-critical systems. We do, however, compare our approaches with those used in other industries in Lessons Learned from Other Industries.

[4]In this, we are distinct from the industry term DevOps, because although we definitely regard infrastructure as code, we have *reliability* as our main focus. Additionally, we are strongly oriented toward removing the necessity for operations—see The Evolution of Automation at Google for more details.

[5]In addition to this great story, she also has a substantial claim to popularizing the term "software engineering."

← PREVIOUS
Foreword

NEXT
Part I - Introduction