This article builds upon Vivek Rau’s chapter “Eliminating Toil” in *Site Reliability Engineering: How Google Runs Production Systems* [1]. We begin by recapping Vivek’s definition of toil and Google’s approach to balancing operational work with engineering project work. The Bigtable SRE case study then presents a concrete example of how one team at Google went about reducing toil. Finally, we leave readers with a series of best practices that should be helpful in reducing toil no matter the size or makeup of the organization.

### SRE’s Approach to Toil

As discussed in depth in the recently published *Site Reliability Engineering*, Google SRE seeks to cap the time engineers spend on operational work at 50%. Because the term *operational work* might be interpreted in a variety of ways, we use a specific word to describe the type of work we seek to minimize: *toil*.

#### Toil Defined

To define toil, let’s start by enumerating what toil is *not*. Toil is not simply equivalent to:

- “Work I don’t like to do”
- Administrative overhead such as team meetings, setting and grading goals, and HR paperwork
- Grungy work, such as cleaning up the entire alerting configuration for your service and to remove clutter

Instead, toil is the kind of work tied to running a production service that tends to be:

- Manual
- Repetitive
- Automatable and not requiring human judgment
- Interrupt-driven and reactive
- Of no enduring value

Work with enduring value leaves a service permanently better, whereas toil is “running fast to stay in the same place.” Toil scales linearly with a service’s size, traffic volume, or user base. Therefore, as a service grows, unchecked toil can quickly spiral to fill 100% of everyone’s time.

As reported by SREs at Google, our top three sources of toil (in descending order) are:

- Interrupts (non-urgent service-related messages and emails)
- On-call (urgent) responses
- Releases and pushes

Toil isn’t always and invariably bad; all SREs (and other types of engineers, for that matter) necessarily have to deal with some amount of toil. But toil becomes toxic when experienced
in large quantities. Among the many reasons why too much
tool is bad, it tends to lead to career stagnation and low morale.
Spending too much time on tool at the expense of time spent
engineering hurts the SRE organization by undermining our
engineering-focused mission, slowing progress and feature
velocity, setting bad precedents, promoting attrition, and causing
breach of faith with new hires who were promised interesting
engineering work.

Addressing Toil through Engineering
Project work undertaken by SREs is key in keeping tool at
manageable levels. Capping operational work at 50% frees up the rest
of SRE time for long-term engineering work that aims
to either reduce tool or add service features. These new features
typically focus on improving reliability, performance, or utilization—efforts which often reduce tool as a second-order effect.

SRE engineering work tends to fall into two categories:

- **Software engineering:** Involves writing or modifying code,
in addition to any associated design and documentation work.
Examples include writing automation scripts, creating tools or
frameworks, adding service features for scalability and reliability,
or modifying infrastructure code to make it more robust.

- **Systems engineering:** Involves configuring production
systems, modifying configurations, or documenting systems
in a way that produces lasting improvements from a one-time
effort. Examples include monitoring setup and updates, load-
balancing configuration, server configuration, tuning of OS
parameters, and load-balancer setup. Systems engineering also
includes consulting on architecture, design, and productioniza-
tion for developer teams.

Engineering work enables the SRE organization to scale up
sublinearly with service size and to manage services more effi-
ciently than either a pure Dev team or a pure Ops team.

Case Study: Bigtable SRE
It's important to understand exactly what tool is, and why it
should be minimized, before engaging boots on the ground to
address it. Here's how one SRE group at Google actively worked
to reduce tool once they realized that it was overburdening the
team.

**Toil in 2012**

In 2012, the SRE team responsible for operating Bigtable, a
Google high performance data storage system, and Colossus,
the distributed file system upon which Bigtable was built, was
suffering from a high rate of operational load.

Early in the year, pages had reached an unsustainable level
(five incidents per standard 12-hour shift; Google purposefully
designs many of its SRE teams to be split across two sites/time
zones to provide optimal coverage without overtaxing on-call engineers with 24-hour shifts), and the team began an effort
to eliminate unnecessary alerts and address true root causes
of pages. With concentrated effort, the team brought the pager
load down to a more sustainable level (around two incidents
per shift). However, incident response was only one component
of the team's true operational load. User requests for quota
changes, configuration changes, performance debugging, and
other operational tasks were accumulating at an ever-increasing
rate. What began as a sustainable support model when Bigtable
SRE was responsible for just a few cells and a handful of cus-
tomers had snowballed into an unpleasant amount of unreward-
ing tool.

The team wasn't performing all of its daily operations “by hand,”
as SREs had created partial automation to assist with a number
of tasks. However, this automation stagnated while both the
size of Google's fleet and the number of services that depend
on Bigtable grew significantly. On any given day, multiple engineers
were involved in handling the tool-driven work that resulted
from on-call incidents and customer requests, which meant
that these SREs couldn't focus on engineering and project work.
In fact, an entire subteam was dedicated to the repetitive but
obligatory task of handling requests for increases and decreases
in Bigtable capacity. To make matters worse, the team was so
overburdened with operational load that they didn't have time to
adequately root cause many of the incidents that triggered pages.
The inability to resolve these foundational problems created a
vicious cycle of ever-increasing operational load.

**Turning Point**

Acknowledging that its operational trajectory was unsustain-
able, the entire Bigtable and Colossus SRE team assembled
to discuss its roadmap and future. While team members were
nearly universally unhappy with the level of operational load,
they also felt a strong responsibility to both support their users
and to make Google's storage system easy to use. They needed a
solution that would benefit all parties involved in the long run,
even if this solution meant making some difficult decisions
about how to proceed in the short term.

After much discussion, Bigtable SRE agreed that continuing
to sacrifice themselves to achieve the short-term goals of their
customers was actually counterproductive, not only the team,
but also to their customers. While fulfilling customer requests
on an as-needed basis might have been temporarily gratifying, it
was not a sustainable strategy. In the long run, customers value
a reliable, predictable interface offered by a healthy team more
than they value a request queue that processes any and every
request, be it standard or an unconventional one-off, in an inde-
terminate amount of time.
Tactics
The team realized that in order to get their operational work under control and improve the Bigtable service for their users, they would have to say “no” to some portion of customer requests for a period of time. The team, supported by management, decided that it was important (and ultimately better for Bigtable users) to respect their colleagues and themselves by pushing back on complex customer requests, performance investigations for customers who were within Bigtable’s promised SLO, and other routine work that yielded nominal value. The team’s management understood that the long-term health of both the team and the service could be substantially improved by making carefully considered short-term sacrifices in service quality. Additionally, they decided to split the team into two shards: one focused on Bigtable, and one focused on Colossus. This split had two advantages: it allowed engineers to specialize technically on a single product, and it allowed the leads of each shard to focus on improving the operational state of a single service.

In addition to temporarily impacting how, and how quickly, they processed user requests, the team recognized that their new focus on reducing operational load would also impact their work in a couple of other key areas: their ability to complete project work and their relationship with partner developer teams. For the time being, SREs would have less bandwidth to collaborate with the core Bigtable development team in designing, qualifying, and deploying new features. Fortunately, the Bigtable developers anticipated that reducing operational load would result in a better, more stable product, and went so far as to allocate some of their engineers to this effort. Assisting the SRE team in improving service automation would ultimately benefit both teams if developers could shorten the window of slowed feature velocity.

The Turnaround Begins: Incremental Progress
Equipped with a narrowed scope and a clear mandate to focus on reducing toil, the Bigtable SRE Team began making progress in clearing their operational backlog. They first turned an eye to routine user requests. The overwhelming majority of requests fell into three buckets:

- Increases and decreases in quota
- Turnups and turndowns of Bigtable footprints
- Turnups and turndowns of datacenters

Rather than trying to engineer an all-encompassing big-bang solution, the team made an important decision: to deliver incremental progress.

Bigtable SRE first focused on fully automating the various footprint- and quota-related requests. While this step didn’t eliminate tickets, it greatly simplified the ticket queue and reduced the amount of time it took to complete requests. The team could now fulfill each request by simply starting automation to complete the task, eliminating the several manual steps previously necessary.

Next, the team focused on wrapping automation into self-service tools. Initially, they simply added quota to an existing footprint, which was both the most common request and the easiest request to transition to self-service. SREs then began adding self-service coverage for more operations, prioritizing according to complexity and frequency. They tackled common and less complex tasks first, moving from quota reductions, to footprint turnups, to footprint turndowns.

Bigtable SRE’s iterative approach was twofold: in addition to tackling lower-hanging fruit first, they approached each self-service task starting from the basics. Rather than trying to create fully robust solutions from the get-go, they launched basic functionality, upon which they incrementally improved. For example, the initial version of the self-service software for quota reductions and footprint turndowns couldn’t handle all possible configurations. Once users were equipped with this basic functionality, the engineers incrementally expanded the self-service coverage to a growing fraction of the request catalog.

End Game
By breaking up the toil problem into smaller surmountable pieces that could deliver incremental value, Bigtable SRE was able to create a snowball of work reduction: each incremental reduction of toil created more engineering time to work on future toil reduction. As shown in Figure 1, by 2014, the team was in a much improved place operationally—they reduced user requests from a peak of more than 2200 requests per quarter in early 2013 to fewer than 400 requests per quarter.
Looking Forward

While Bigtable SRE significantly improved its handle on toil, the war against toil is never over. As Bigtable continues to add new features, and its number of customers and datacenters continues to grow, Bigtable SRE is constantly on the offensive in combating creeping levels of toil. Perhaps the most significant change Bigtable SRE underwent in this process was a shift in culture. Before the turnaround, the team viewed operational work as an unpleasant but necessary task that they didn’t have the power to refuse or delay. Since the turnaround, the team is extremely skeptical of any feature or process that will add operational work. As team members challenge and hold each other accountable for the level of operational load on the team, they aim to never regress to similarly undesirable levels of toil.

Best Practices for Reducing Toil

Now that we’ve seen how one particular SRE team at Google tackled toil, what lessons and best practices can you glean from a massive-scale operation like Google that apply to your own company or organization?

As they’re tasked with running the entire gamut of services that make up Google production, SRE teams at Google are necessarily varied, as are their approaches to toil reduction. While some of the particular approaches taken by a team like Bigtable SRE might not be relevant across the board, we’ve boiled down SRE’s diverse approaches to reducing toil into some essential best practices. These recommendations hold regardless of whether you’re approaching service management from scratch or looking to help a team already burdened by excessive toil.

Buy-in Is Key

As demonstrated by the Bigtable SRE case study, you can’t tackle toil in a meaningful way without managerial support behind the idea that toil reduction is a worthwhile goal. Sometimes long-term wins come with the tradeoff of short-term compromises, and securing managerial buy-in for temporarily pushing back on routine but important work is likely easier said than done. The key here is for management to consider what measures will enable a team to be significantly more effective in the long run. For example, Bigtable SRE was only able to rein in the toil overwhelming their team by deprioritizing feature development and manual and time-consuming customer requests in the short term.

Bigtable SRE also found that breaking down toil reduction efforts into a series of small projects was key for a few reasons. Perhaps most obviously, this incremental approach gives the team a sense of momentum early on as it meets goals. It also enables managers to evaluate a project’s direction and provide course corrections. Finally, it makes progress easily visible, increasing buy-in from external stakeholders and leadership.

Minimize Unique Requirements

Using the “pets vs. cattle” analogy discussed in a 2013 UK Register article [2], your systems should be automated, easily interchangeable, replaceable, and low-maintenance (cattle); they should not have unique requirements for human care and attention (pets). Should disaster strike, you’ll be in a much better position if you’ve created systems that can be recreated easily from scratch. Tempting as it might be to manually cater to individual users or customers, such a model is not scalable.

Similarly, understand the difference between parts of the system that require individual care and attention from a human versus parts that are unremarkable and just need to self-heal or be replaced automatically. Depending on your scale, these components might be hosts, racks of hosts, network links, or even entire clusters.

Be thoughtful about how you handle configuration management. By using a centrally controlled tool like Puppet, you gain scalability, consistency, reliability, reproducibility, and change management control over your entire system, allowing you to spin up new instances on demand or push changes en masse.

While many people and teams recognize that building one-off solutions is suboptimal, it’s still often tempting to build such systems. Actually steering away from creating special cases for short-term efficacy and insisting on standardized, homogeneous solutions requires focus and periodic review by team leads and managers.

Invest in Build/Test/Release Infrastructure Early

Instituting standardization and automation might be a hard sell early on in a service’s life cycle, but it will pay off many times over down the road. Implementing this infrastructure is much harder later on, both technically and organizationally.

That said, there’s a balance between insisting on this approach wholesale, thus hurting velocity, versus postponing infrastructure development until suboptimally late in the development cycle. Try to plan accordingly—once you’re beyond the rapid launch-and-iterate phase and relatively certain that the system will have the longevity to warrant this kind of investment, put sufficient time and effort into developing build, test, and release infrastructure.

Audit Your Monitoring Regularly

Establish a regular feedback loop to evaluate signal versus noise. For each real-time alert, repeat the mantra, “What does a human being need to do, right this second?” The Site Reliability Engineering chapter “Monitoring Distributed Systems” covers this topic in depth.
Invent More, Toil Less

**Conduct Postmortems**

The need for postmortems may not surface in the course of everyday work, but consistently undertaking them massively contributes to the stability of a system or service. Instead of just scrambling to get the system back up and running every time an incident occurs, take the time to identify and triage the root cause after the immediate crisis is resolved. As detailed in the SRE chapter “Postmortem Culture: Learning from Failure,” these collaborative postmortem documents should be both blameless and actionable. Avoid one-size-fits-all approaches: this exercise should be lightweight for small and simple incidents but much more in-depth for large and complex outages.

**No Haunted Graveyards**

Even when it comes to companies and teams that consider themselves fast-moving and open to risk, parts of production or the codebase are sometimes considered “too risky” to change—either very few people understand these components or they were designed in such a way that there’s a risk assigned to changing or touching them. Our goal is to control trouble, not to avoid it at all costs. In such cases of perceived risk, smoke out risk rather than leaving it to fester.

**Conclusion**

Any team tasked with operational work will necessarily be burdened with some degree of toil. While toil can never be completely eliminated, it can and should be thoughtfully mitigated in order to ensure the long-term health of the team responsible for this work. When operational work is left unchecked, it naturally grows over time to consume 100% of a team’s resources. Engineers and teams performing an SRE or DevOps role owe it to themselves to focus relentlessly on reducing toil—not as a luxury, but as a necessity for survival.

The type of engineering work generated by toil reduction projects is much more interesting and fulfilling than operational work, and it leads to career growth and healthier team dynamics. Google SRE teams have found that working from the set of best practices above, in addition to constantly reassessing our workload and strategies, has equipped us to continually scale up the creative challenges, business impact, and technical sophistication of the SRE job.

**Acknowledgments**

Special thanks to the following for helping with background for this article: Olivier Ansaldi, Brent Chapman, Neil Crellin, Sandy Jensen, Jeremy Katz, Thomas Labatte, Lisa Lund, and Nir Tarcic.

**References**
