# SRE in transition: from startup to established business

Laura de Vesine
Datadog, Inc.

DATADOG

## Life of a Startup

Ship or Die

Avoid fancy technology: you don't need it, and it's slow to use

Probably no SLOs, no incident reviews

SRE's job is to enable shipping faster any way you can

Startups take on debt to grow, both tech and money

*Credit to Emil Stolarsky's "Unified Theory of SRE"*



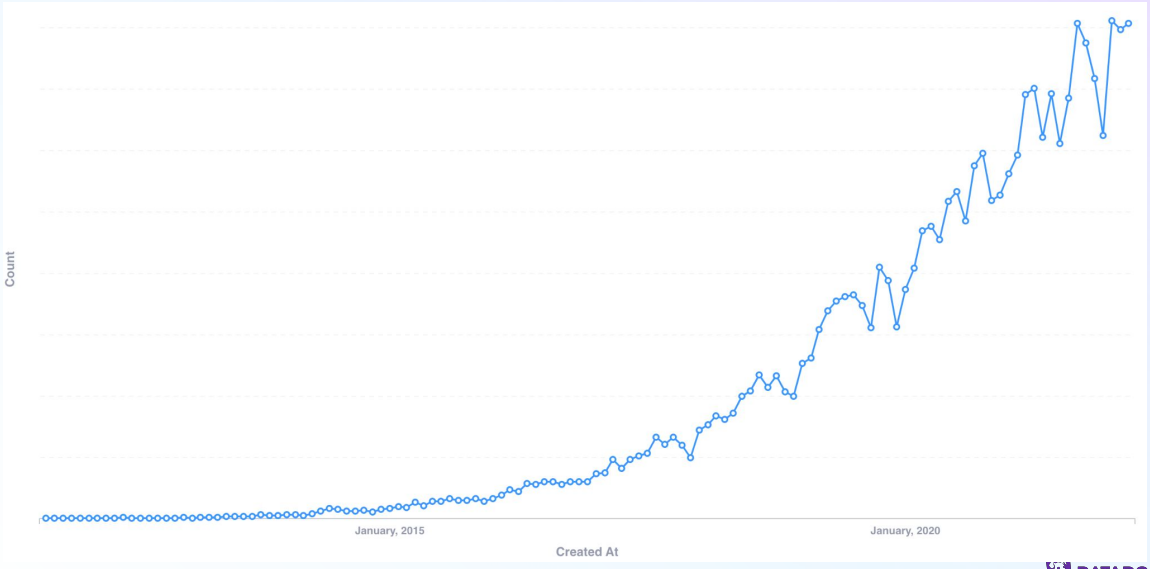
Startups are companies that are "dead" quite soon if they don't ship – the rule is, ship or die (aka being default dead). That means that, if you *have* an SRE team at a startup, they need to be focused on the same mission as everyone else – get a product out the door ASAP. You probably don't have all that "fancy" stuff like incident reviews and SLOs, and you shouldn't use the finicky cool technology. The big goal is to get to sustainability before you run through all your investor money (and before you collapse under your own tech debt).

Unified Theory of SRE:
https://www.usenix.org/conference/srecon22emea/presentation/stolarsky
Rose photo by feey on Unsplash
Burning money photo by Jp Valery on Unsplash


For many startups, the goal is to get to that moment when suddenly… it's all a success. Growth takes off, and you often hit a hypergrowth phase as part of moving from default-dead to default-alive.

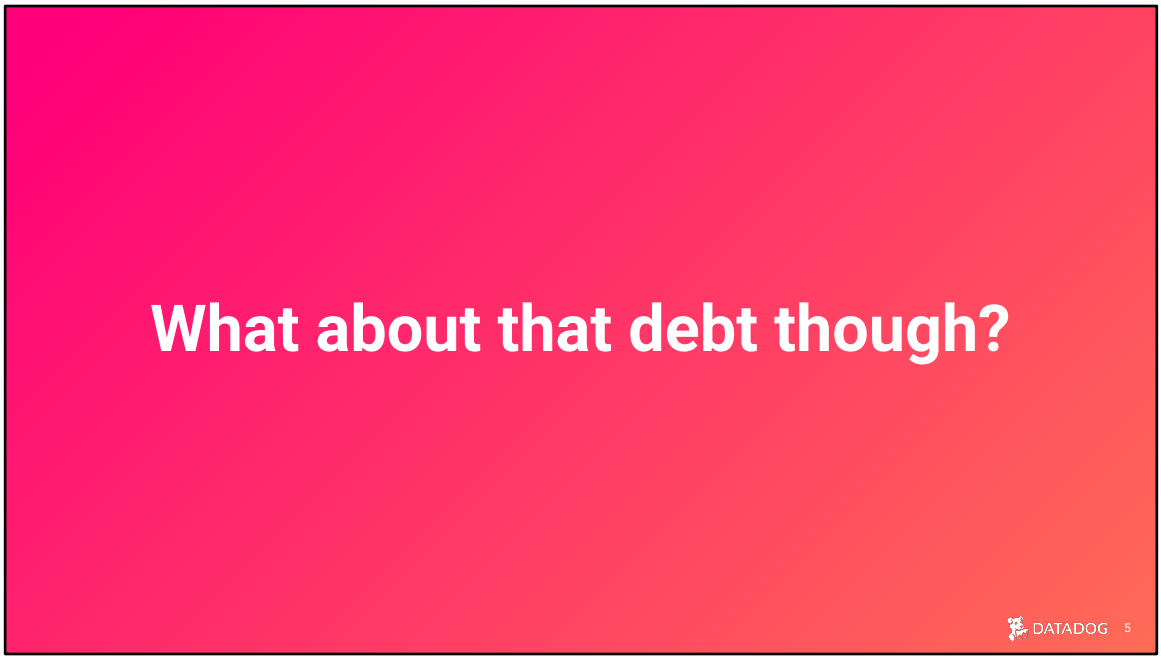
Photo by Pixabay: https://pixabay.com/photos/rocket-launch-rocket-take-off-67723/

# Datadog: Commits Over Time




DATADOG 4

Here's an example of what hypergrowth can look like. This is a graph of commits over time at Datadog, but I could generate graphs with a similar shape from all sorts of metrics (this one was just easy). You can see a long, slow phase (those aren't 0, they're just comparatively very low) and then this takeoff point. That's hypergrowth post-startup.



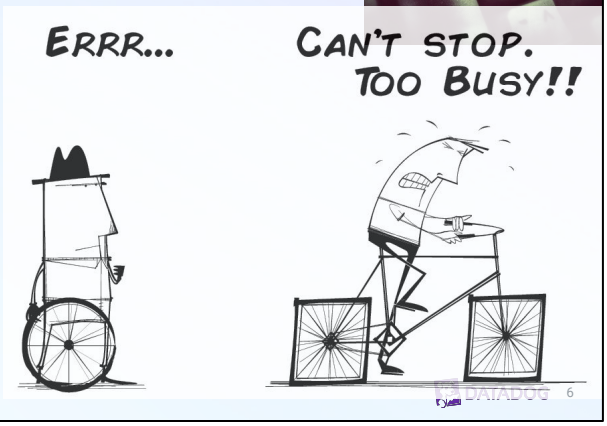

I glossed over this a bit, but a defining characteristic of startups is taking on various kinds of debt in order to fuel that takeoff point. So, what kinds of debt are we talking about here?

## Tech Debt

Lack of scaling ability

Lack of shared infrastructure

“Basic” systems engineering principles not at all universal





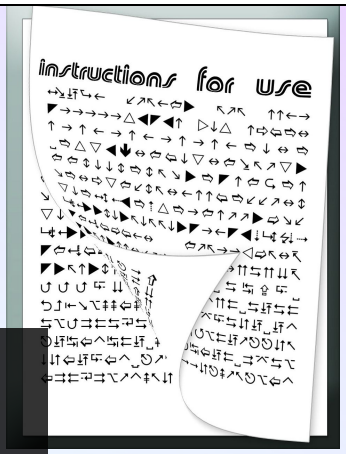

What does that tech debt look like? It can be a lot of things, from features that can’t take 10x, let alone 100x traffic without falling over, to a total lack of API boundaries meaning 1500 developers work on the same single monolithic service and no one is quite sure who owns what. It can mean that team A over here uses K8s and envoy for service discovery, and team B over here is using raw EC2 instances and hard-coded IP addresses for service discovery. It probably means a lot of the “basic” systems engineering that we tend to focus on as SREs aren’t widespread (or are completely lacking), so you see systems with things like infinite retries with no backoff, timeouts set to 15 minute on calls that normally take 500ms, services that require full-mesh client connections by design, traffic sharding that’s hard-coded and risky to change…. Again, I want to emphasize that the company was *right* to take on this debt in order to get off the ground – but now that we’re here, we have to pay interest on this debt and, ideally, start to pay it down

Square bicycle image from Alan O’Rourke http://audiencestack.com/static/blog.html
Grungy keyboard via Pixabay
https://pixabay.com/photos/keyboard-grunge-antique-computer-509465/
Old monitors image via Pixabay
https://pixabay.com/photos/computers-monitors-teams-cables-pc-814257/

# Manual Operations

Manual operations everywhere

“Feeding human blood to the machines”




DATADOG 7

Historically, human time was cheaper than building and running automation. As a result, there’s manual operational work as a system foundation *everywhere*: to scale up services, to change which job processes which data/sharding, handling and blocking abusive customer traffic, integration testing, deployments… Manual operations are perceived as “normal” and even “safer” or “preferred” over automation. But the opportunity cost to your humans rises (including as you have more humans and services), and this becomes an unsustainable form of debt as well

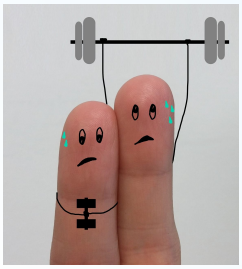
User manual via Pixabay
https://pixabay.com/illustrations/instructions-user-manual-76729/
Gear image via Pixabay
https://pixabay.com/photos/a-people-evertebrat-side-view-3151524/
Hand crush sign public domain:
https://publicdomainvectors.org/en/free-clipart/Mechanical-crush-hazard-warning-sign-vector-image/15177.html

## Oncall and Ops Challenges

Cognitive Load

Oncall Burnout

Lack of incident review




DATADOG 8

Every engineer is expected to (and needs to) know the whole stack, including service discovery and machine specifics, to build or debug anything (“you build it/you run it” and lack of platforms and uniformity). It’s common to page a human for problems that are affecting a single customer and may even self resolve, both because of the preference for manual operations and because single customers meant a lot more when you were a startup. Oncall rotations may be too small to be long-term sustainable as well. Incidents aren’t reviewed, which means we don’t look at the underlying causes of our problems – we “fix the bug and move on”, allowing our systems to become less stable over time.

Weight lifting images from Pixabay
https://pixabay.com/illustrations/weightlifting-fatigue-sweat-gym-1872377/ and
https://pixabay.com/illustrations/gym-weights-toil-limits-effort-2020303/
Pager beep icon created by Freepik - Flaticon
https://www.flaticon.com/free-icons/beep
Multitasking man via Pixabay
https://pixabay.com/illustrations/multitasking-efficiency-2840792/

## Cultural Friction

"Process" is viewed as inherently negative

Work optimization is highly localized




DATADOG 9

Many employees will view any "process" as extremely negative – for example, "being required to write and present a postmortem feels like punishment for having an outage; I'd rather just fix the problem and move on." Teams are also unaccustomed to thinking about (or being subject to) global cost optimization rather than for their own team. They resist migrations because they make the individual team less efficient, and teams *requesting* migrations rarely think about the large, distributed costs of requiring every team to migrate themselves.

Man with banana from Pixabay:https://pixabay.com/photos/suit-business-man-business-man-673697/
"Rules" from Pixabay:
https://pixabay.com/photos/office-employees-rules-consultation-4249395/
"No Jumping" from Pixabay:
https://pixabay.com/photos/sign-ordinance-law-bridge-legal-14089/
"Trust the process" motivation sticker created by Karacis - Flaticon
https://www.flaticon.com/free-stickers/motivation


Well! We're SREs, obviously all that stuff above is bad, and now that we're *not* in ship-or-die mode… I want to fix it!

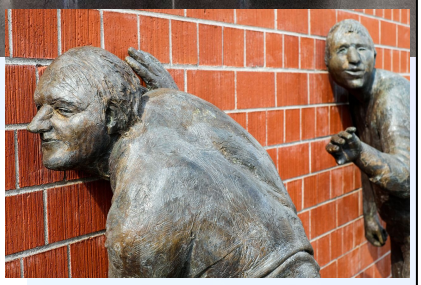
## Establish Credibility

Build trust and context

At all costs avoid the phrase "at Google we..."

Do your devs' job! (It's okay if you're bad at it)

"Sell" to management



DATADOG 11

*Before you try to fix anything*, you need to establish credibility. Why should anyone listen to you? And "I was at <bigco>" is not a good reason – something that worked at bigco wouldn't have worked when the company was a startup; why would it work now? So, as an individual and a team, how should you establish credibility as a person worth listening to about making things better?

The first thing you need to do is *listen*. Understand why things were built the way they are, and why people are doing things the way they are right now. *This should be blameless, the same way a postmortem would be*. Work from the assumption that decisions made in the past were the best ones in that moment, even if now we should move on from them and do things differently. Make that blamelessness *excruciatingly clear* when you ask people questions: "why is it that way" can be judgmental if you just ask that. Ask things like "can you help me understand the context of this design?" and "do you know the limitations that led to <design choice>?" instead.

Take some time to actually *do the work* you want to make changes around – join your dev teams for a bit and try to actually do their jobs to understand what's hard, what's easy, and how you might make it better. Don't expect to be good at it… it's okay to say publicly that you struggled with the work. Validating that their job is *hard* is actually valuable.

You also need to sell your vision of the world to management. Why will doing things your way make the bottom line better? This might look like getting information on the opportunity costs or reliability challenges of existing tech debt and infrastructure, or existing operational or oncall practices. It could look like explaining to management why a new process will give them more visibility for less time. Think about what the advantage is for the company as a whole when individual engineers are better

supported.

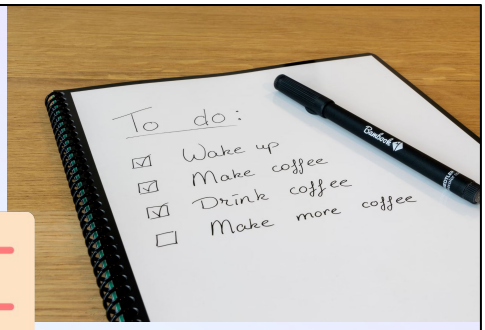
Mugs image courtesy of Ian Nowland (Datadog SVP)
Trust tiles photo by Ronda Dorsey on Unsplash
Listening statues via Pixabay
https://pixabay.com/photos/sculptures-bronze-listen-to-figures-2209152/
Business tree via Pixabay
https://pixabay.com/illustrations/business-tree-growth-success-team-1137366/

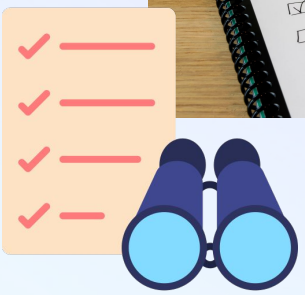
## Scope Your Work

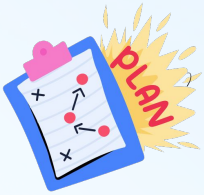
There's a lot of problems. Make a plan

Cluster problems by underlying causes

Only do what you can





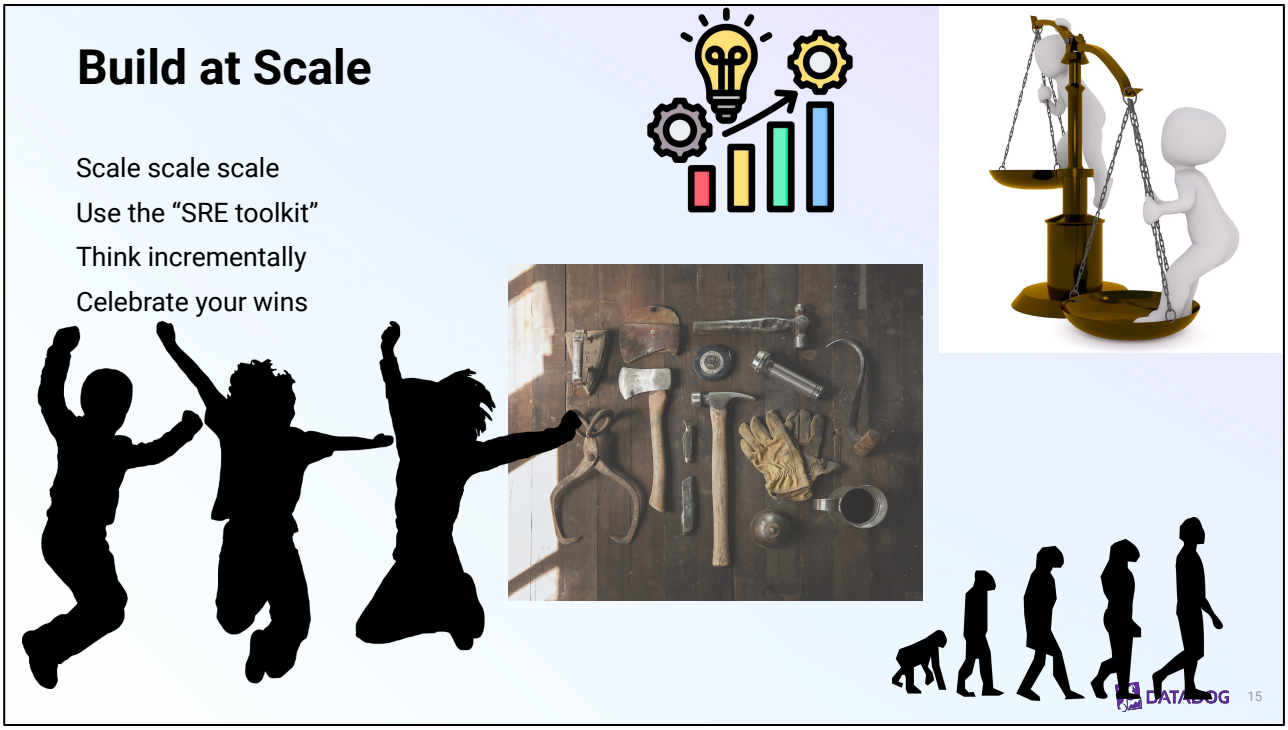

In any organization, there's an infinite amount of "make it run better" work that might be done. When we want to transition our SRE team from startup mode to "srs bzns" mode, there's probably about 2 dozen things you really wish you could fix right now. Obviously you can't do that, so let's make a plan. Generally, it works well to prioritize where the current pain points are felt most acutely: either in eng satisfaction, or reliability, or both. Prioritizing these will (among other things) help you build that credibility you need to keep solving problems. Some examples might be single-team focused ("your database team is being asked to provide SQL-query level access to the entire, company-wide database instance with 100% uptime", or "every time a customer sends us more traffic we have a 2-hour incident requiring a team to reshard their service manually"), or "small" issues relating to many teams, like "standards for how quickly to respond to a page vary wildly, and are generating significant friction" or "we routinely page for single-customer small impacts".

Once you have some thoughts on specific problems you'd like to address first, cluster them by underlying causes just like you would for a postmortem – for example, "failure is not normal" for 100% DB uptime or paging for every customer issue. Other example clusters might be "engineers are using <infrastructure> badly", "<infrastructure> has major stability or scaling challenges", "default practices result in badly-scaling systems", "lack of deep investigation into our outages", "we're missing a tool (or have 5 different tools) to do <important distributed system thing>"

You want to address the problems you see by root cause/cluster, but you're going to have to prioritize. Choose a number of things to work on (which might be one!) that your SRE org actually has the bandwidth for. You do no one a favor by doing too many things at once.

Blurry focus photo by Stefan Cosma on Unsplash
Todo list photo by Thomas Bormans on Unsplash
Dart sticker created by kerismaker - Flaticon
https://www.flaticon.com/free-stickers/dart
Scope icon created by Muhammad Atif - Flaticon
https://www.flaticon.com/free-icons/scope
Strategic plan sticker created by smashingstocks - Flaticon
https://www.flaticon.com/free-stickers/strategic-plan

focus



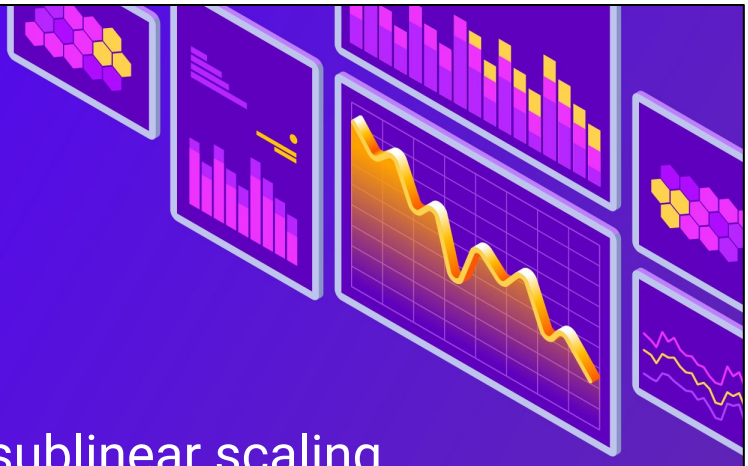

Okay, so let’s fix stuff! Chances are, you’re late to the party and the company’s already been in hypergrowth for a while. And there’s going to be a lot more engineers doing things “wrong” than you have SREs to tell them how to do it “right”. So like every SRE team ever, you need to think about scale – in this case, how do you scale your efforts at cultural and technological change?

If the problem is technological, you might want to build (or start using) a specific piece of infrastructure or technology. You’re going to need to help teams migrate to it, not just demand that they do – again, you have to sell this. If the problem is cultural, you might build process or make changes to company onboarding (or do presentations and have conversations with colleagues) to start to change people’s thinking. In all cases, *don’t* plan to do all the work yourself – think about how to build self-sustaining solutions, and “natural” champions for your changes. One thing to *avoid* is trying to teach deep expertise to a few people at a time and then hope they’ll transmit it to their org. That’s not going to scale fast enough in hypergrowth, and the people you build up that way are unlikely to be rewarded for that time, either. Think more about small champions – people who will see a problem, think “oh, I heard SLOs could solve this” and say that to their team (who might then reach out to SRE for more detailed help).

Speaking of SLOs, the “SRE toolkit” is your friend here. That’s things like SLOs, incident reviews and postmortems, production readiness reviews, structured and limited oncall and ops expectations, the reflex to automate… We built that toolkit for a reason. They can help address underlying cultural and technological problems that are causing pain. But don’t apply the ritual blindly. Think about what problem you’re solving, and how the tool solves it.

Think incrementally – “big bang” solutions take forever to show value, and you’ll probably get the needs of the organization wrong. How can you build technology that improves the status quo in a quarter, or even a couple of days? For example, a tool that moves from ignoring your vacation time when scheduling oncall to one that respects it can make a big difference. How can you convince one person to do things differently, and have them share that with their team, and more broadly?
And finally, celebrate your wins. Large-scale change is hard and slow; don’t let yourselves lose sight of the progress you make.

Scale image via Pixabay
https://pixabay.com/illustrations/libra-pan-weigh-kitchen-scale-2071315/
Jumping people via Pixabay
https://pixabay.com/vectors/silhouette-dancing-jumping-people-3095150/
Tools photo by Todd Quackenbush on Unsplash
Trend icon created by Uniconlabs - Flaticon https://www.flaticon.com/free-icons/trend
Evolution silhouettes via Pixabay
https://pixabay.com/vectors/evolution-walking-charles-darwin-297234/



SRE is a goal: sublinear scaling
And a set of ways to get there
*Not* a dogma to implement for its own sake