200 XP

Key SRE principles and practices: virtuous cycles

10 minutes

If it is really true that in some sense “you are what you do”, then we’ve come to the heart of this module. In this unit we are going to look at two of the practices that are often considered core to the practice of SRE. Both originate from the principle that it is important to create “virtuous cycles." Virtuous cycles in this context are practices that build feedback loops in an organization that help that organization continuously get better. There will be entire follow-on modules on exactly these two practices so we’re only going to skim the surface with an overview of each here.

Virtuous cycle #1: SLIs and SLOs

Earlier in this module, we made a big deal about working towards the “appropriate level of reliability”. This is precisely the place where that concept gets brought to bear.

Let’s say you have a new service you are planning to bring to production (either one that has been constructed or one that is still in the planning process). As part of that process, it is important to make some decisions about its desired reliability. If you are not writing all of the code yourself, these decisions are made (and this is crucial) in collaboration with the developers making the thing.

The first decision that gets made is what will be used as indicators of the service’s health (a Service Level Indicator or SLI). Another way to ask this will be “How do you know when it's up/working well?”. There are lots of ways to track this which we'll explore in detail later, but these things are typically success vs. failure measures (does the service successfully complete an operation some percentage of the time), measures of timing (did we return an answer within a certain threshold of time), measures of throughput (did we process a certain amount of data) or combinations of all of these. For a simple example, we might say an SLI for our service is how often it returned success, indicated via an HTTP 200 code (vs. a 500 or some other code).

Now that we have a clear indicator for how to tell how the service is doing, we’re going to want to decide what level of reliability we expect or desire from it. For example, do we expect over a period of a day that we'll see a failure rate of 20% from the service? We're using round and large numbers here because they're easy to reason about in the beginning. In later modules, we’ll increase the complexity and precision of statements like this (“which users will see that error rate? some of them? most of them?” and so on). That expectation, created in collaboration with the service’s developer, is a Service Level Objective (SLO).

The SLO needs to be something that can be accurately measured and represented in your monitoring system. It's meant to be an objective, well understood goal for the reliability for the service—what is the number that is good enough? There's no "well, I think the service has been doing ok for the last week or so, but it's kind of hard to tell" going on here. Either the service is meeting its SLO or it isn't, the data should be clear. If it isn't meeting its SLO (especially repeatedly over a span of time), then something is wrong and needs to be addressed.

Error budgets

It can be straightforward to understand that an organization might snap into action if a service doesn't meet its SLO, but SRE takes this whole concept another step forward for the cases where the SLO is being met or exceeded. Some organizations use SLOs to construct what they call "error budgets".

To demonstrate this idea, lets use the sample service we've been discussing and its SLO of 80% success (think of it as "must be up 80% of the time"). With the SLO of 80% uptime we've declared that our service must be up 80% of time. But what about the other 20%? If our service is down that other 20%, we don't really "care" because we've decided being up that extra 20% isn't important to us as a service goal.

So if we don't care about what happens during that time, what can we do with the service? One thing we can certainly do is perturb the running service by upgrading it, perhaps with a new release that adds some features. If that new release stays up and doesn't add any downtime, great. If that new release causes the service to be less stable and return errors another 10% of time as it gets debugged, still just fine. We're within our budget of allowed unreliability.

An error budget is the difference between the service's potential perfect reliability and its desired reliability (100% - 80% = 20%). In this case, the error budget gives us a fund of 20% unreliability—20% time where we "don't care whether it is up or not because it will still be in spec". We can draw on and spend that 20% time any way we'd like (perhaps with more releases) until it is exhausted just like any other budget.

Error budgets are also used in some organizations for the less happy case, the one where you aren't making your SLO. In that case, you might choose to do something a little more stringent than just "take an action" as mentioned above. Let's say our service has been having some issues and has been up just 60% of the time as indicated by the SLI we chose earlier. We didn't make our objective (the SLO). Our service has used up its error budget. The organization may choose to hold back on a planned release because it knows that perturbing the system even further at this point is likely to only further worsen the reliability situation. Error budgets are usually calculated for a set period of time--a month, quarter, and so on--or on a rolling basis so eventually if the service reliability improves, that budget returns.

During this time of gated releases, the organization might choose to pivot some engineering resources away from feature development towards reliability work to help uncover and improve the source of the problems that caused the service to blow its SLO.

The reason why we say “some organizations” when it comes to error budgets is their implementation, especially in the case where it's used to gate releases, requires a certain institutional buy-in. The organization has to be willing to say that when faced with a release decision, if the reliability of the service to date hasn’t been up to snuff, that release will be held back. That’s not a decision all organizations are willing to make. This is also not the only possible response to a depleted error budget, but it is the one most talked about.

We’ll be talking in considerably more detail about SLIs, SLOs, and error budgets in a separate module, but it is worthwhile highlighting the virtuous cycle aspect of these practices. In theory, it provides a way for an organization to describe, communicate, and act on the reliability of a service in a way that gets everyone working towards better reliability. This feedback loop can be incredibly important.

Virtuous cycle #2: blameless postmortems

The idea of a postmortem—the retrospective analysis of a significant, usually undesired event—is not even remotely specific to site reliability engineering. It isn't even uncommon to the operations world. One thing that is closer to being distinctive is SRE’s insistence that postmortems need to be “blameless.” They need to focus on the failure of the process or the technology during the incident, not the actions of specific people. “What was it about the process we had in place that allowed X to do the thing that led to the failure? What information did that person not have available, or even prominent at the moment that led to them coming to the wrong conclusion? What sort of guardrails should have been in place so it wasn’t possible to have such a catastrophic failure take place?” These are the sort of questions that are asked in a blameless postmortem.

The tenor and direction of these questions is crucial. They are searching for ways to improve the systems or processes, not ways to punish the individuals whose use of those systems or processes in good faith contributed to the outage. It's important to remember “You can’t fire your way to reliable”. In our experience, an organization that fires a person every time there is a production incident (with very few exceptions), will not learn from those incidents. Instead, there will be a single individual left, shaking in the corner, afraid to make any changes to anything at all.

But a well functioning post-mortem process in an organization creates a virtuous cycle. It lets the organization learn from its outages and continuously improve its systems (providing proper analysis and followup is done).

This relationship to failures and errors—embraced by the organization as opportunities for learning and improvement—is a core principle of site reliability engineering. The construction of virtuous cycles to make use of these opportunities and to guide the organization towards greater reliability is another.

Let's explore some other principles and practices, those centered on the human side of SRE, in our next unit.

Check your knowledge

1. What does SLI stand for (in an SRE context)?

- ○ Standard Level Indicator
- ◉ Service Level Indicator ✓
- ○ Safe Load Indicator
- ○ System Level Interface

2. What does SLO stand for (in an SRE context)?

- ○ Service Level Outcome
- ○ Standard Line Operations
- ○ Service Load Objective
- ◉ Service Level Objective ✓

3. If you exhaust your error budget for a service, what should you do?

- ○ Immediately restart the service involved so it can be brought back into compliance
- ○ Put a hold on further releases of that software until the service has returned to the agreed upon level of reliability
- ○ Immediately redirect engineering resources associated with this service from feature development to fixing the reliability problem
- ◉ React in the service- or organizationally-specific way previously agreed upon when creating that error budget ✓

When setting up an error budget for a service, it is important to decide (and document) how the organization will respond should it be exhausted. There are many possible responses, you need to choose and stick to the ones that work best in your particular context. An error budget is a way of tracking the aggregate reliability

of a service over time. It is important to accompany this with a shared understanding of how the organization will react should the amount of acceptable unreliability be exceeded. There are many possible responses, you need to choose and stick to the ones that work best in your particular context. There is no one right reaction for every one and every situation.

4. If you exceed your error budget for a service, what should you do?

- ○ Speed up the release cadence for that service
- ○ Focus more on feature development for that service and deploy those features faster
- ◉ It depends ✓

  There are many ways to "spend" an excess error budget. Increasing the release cadency or feature velocity, deploying more beta features, conducting more disruptive testing, and so on. It is also fine (though this comes with its own set of problems) to do nothing and simply operate at a greater level of reliability. All of these responses are situation specific and should be decided in your specific context.

5. When a downtime or other incident occurs, should you immediately terminate the people involved?

- ○ Yes, the person or people who made the mistake should always take responsibility for their actions and pay the consequences
- ◉ No, except under certain rare and extraordinary circumstances ✓

  Feedback loops like blameless postmortems are crucial to continuous improvement. If an organization learns from an incident instead of firing the people involved (the people who may understand best what happened), it can work directly on the reliability of its services, systems and products.

Next unit: Key SRE principles and practices: The human side of SRE

Continue >